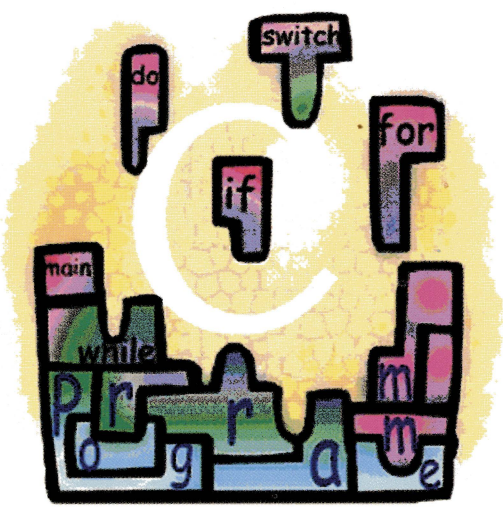
APPRENDRE À PROGRAMMER PAR L'EXEMPLE

Fiches de cours

et exercices corrigés en C

Adnane ZRIBI **Zied ELOUEDI**



Centre de Publication Universitaire

Apprendre à programmer par l'exemple

Fiches de cours et exercices corrigés en C

Adnane Zribi
Maître-Assistant
ISG de Tunis

Zied Elouedi
Maître-Assistant
ISG de Tunis

Centre de Publication Universitaire
2006

Tous droits de reproduction, de traduction et d'adaptation réservés pour tous pays.
© Centre de Publication Universitaire,Tunis, 2006.
B.P: 255 TUNIS-CEDEX 1080 - TUNISIE
Tél: (216) 71 874 426 Fax: (216) 71 871 677

Table des matières

Avant-propos **1**

Fiches de cours **3**
Fiche N° 1 : Qu'est-ce qu'un programme ? 5
Fiche N° 2 : Qu'est-ce qu'un compilateur ? 6
Fiche N° 3 : Qu'est-ce qu'une variable ? 7
Fiche N° 4 : Les tableaux 8
Fiche N° 5 : Les tableaux multidimentionnels 10
Fiche N° 6 : Les chaînes de caractères 11
Fiche N° 7 : Structure de contrôle : Condition simple 13
Fiche N° 8 : Structure de contrôle : Condition multiple 14
Fiche N° 9 : Structure de contrôle : Itération 15
Fiche N°10 : La boucle "for" 16
Fiche N°11 : Les boucles "while" et "do...while" 17
Fiche N°12 : Les fonctions 18
Fiche N°13 : Les paramètres 20
Fiche N°14 : La récursivité 22

Énoncés **23**
Série N° 1 25
Série N° 2 27
Série N° 3 29
Série N° 4 33
Série N° 5 37
Série N° 6 39
Série N° 7 41
Série N° 8 43
Série N° 9 47
Série N° 10 49
Série N° 11 55

Corrigés..... **57**
Série N° 1 59
Série N° 2 63
Série N° 3 69
Série N° 4 85
Série N° 5 93
Série N° 6 111
Série N° 7 119
Série N° 8 129
Série N° 9 135
Série N° 10 141
Série N° 11 165

Avant-propos

« *Je comprends bien en cours, mais au moment de faire les exercices, je me trouve bloqué !* » Combien de fois avons-nous reçu cette complainte ou d'autres du même ordre de la part de nos étudiants.

C'est qu'à l'instar des mathématiques par exemple, où la démonstration des théorèmes relève plus de l'exercice et de l'expérience que de la théorie et de l'abstraction, la programmation ne peut se résumer en un ensemble de notions théoriques. En fait, le nombre de notions à assimiler est très limité et leur compréhension est relativement aisée. Mais la résolution de chaque problème reste différente et la solution à apporter à chaque cas est toujours spécifique.

Une autre analogie nous semble adaptée. C'est celle du jeu de construction pour enfants « Lego ». Les pièces sont de seulement quatre ou cinq types et couleurs. Il s'agit pour l'enfant de construire des édifices toujours différents au gré de sa créativité et de sa fantaisie. Bien sûr qu'il y a des règles, mais ce sont généralement celles du bon sens. Et surtout, ces règles ne s'acquièrent et ne sont assimilées par l'enfant que par l'expérience.

Des livres donnant les éléments théoriques de la programmation des ordinateurs, il en faut et il y en a, à foison. Le besoin se fait sentir, pensons-nous, d'une collection d'exercices corrigés et de cas de programmation résolus qui aident l'apprenti programmeur à progresser ; avec la possibilité de consulter des fiches de cours sommaires portant sur les notions de base de la programmation.

Cette collection d'exercices, nous l'avons voulue progressive, la difficulté y va croissante. Nous l'avons voulue accessible, les exercices y sont organisés par thème. Nous l'avons voulue exhaustive, mais, le sage Conficius l'a dit, « *si j'avais voulu que ce travail soit parfait, je ne l'aurais jamais terminé.* ».

Cette collection d'exercices corrigés est destinée à tous les débutants en programmation. Elle résulte d'une expérience de plusieurs années d'enseignement de la programmation. Le langage que nous avons choisi est le C. Mais à ce stade de la programmation, nous avançons que tous les langages se valent et, de toute manière, le passage de l'un à l'autre est extrêmement aisé.

Nous formulons l'espoir que cet ouvrage puisse, tout en accompagnant les apprenants, les initier aux joies et aux plaisirs de la programmation.

FICHES DE COURS

Fiche N° 1
Qu'est-ce qu'un programme ?

Définition 1 :

Un programme est une suite d'instructions élémentaires destinées à être exécutées par un ordinateur afin d'effectuer une tâche précise.

Définition 2 :

Un programme est une liste ou une séquence d'instructions qui, exécutées par un ordinateur, permettent d'aboutir à un résultat à partir d'un ensemble de données. Ces instructions sont codées dans un langage de programmation (C, PASCAL, ADA, ...).

Pour construire un programme, on peut avoir besoin d'écrire un algorithme. Il s'agit en fait d'un programme mais écrit dans un langage intermédiaire plus ou moins formel appelé langage algorithmique.

Fiche N° 2
Qu'est-ce qu'un compilateur ?

Définition :

Un compilateur est un programme qui permet de traduire un programme écrit dans un langage évolué en un programme en langage machine (seul langage compréhensible par le microprocesseur). À chaque langage de programmation correspond un compilateur différent. On a ainsi le compilateur Pascal, le compilateur C, etc.

À l'occasion de cette opération de traduction, le compilateur permet de découvrir les erreurs de programmation qui éventuellement auront été commises. Ces erreurs peuvent être d'ordre lexical, syntaxique ou même logique.

Fiche N° 3
Qu'est-ce qu'une variable ?

Définition 1 :

Une variable est un emplacement de mémoire qui, au cours de l'exécution du programme, peut recevoir différentes valeurs d'un même type.

Définition 2 :

Élément essentiel de la programmation, les variables permettent de stocker des informations à tout moment de l'exécution d'un programme et de les réutiliser à n'importe quel autre moment.

Définition 3 :

Emplacement de stockage nommé qui peut contenir des données pouvant être modifiées pendant l'exécution du programme. Chaque variable a un nom qui l'identifie de façon unique. Les noms de variables doivent satisfaire à un ensemble de critères qui dépendent du langage de programmation choisi.

Fiche N° 4
Les tableaux

Définition :

Un tableau est une variable composée. C'est un ensemble fini d'éléments de même type, stockés en mémoire à des adresses contiguës.

Déclaration :

La déclaration d'un tableau en C se fait comme suit :

```
type NomDuTableau [NombreElements];
```

où

`NombreElements` est une constante entière positive.

Par exemple, la déclaration :

```
int Tab [10];
```

indique que **Tab** est un tableau de 10 éléments de type int. Cette déclaration alloue donc en mémoire pour la variable **Tab** un espace suffisant pour stocker 10 entiers.

Accès :

On accède à un élément du tableau en utilisant l'opérateur [ ] avec l'indice de l'élément voulu. Les éléments d'un tableau sont toujours numérotés de 0 à NombreElements diminué de 1. Par exemple, le programme suivant remplit puis affiche trois éléments du tableau **Tab** :

```
void main ()
{
   int Tab [10];
   Tab [0] = 3; Tab [1] = 5; Tab [2] = 8;
   printf ("%d\n", Tab [0]);
   printf ("%d\n", Tab [1]);
   printf ("%d\n", Tab [2]);
}
```

On peut initialiser un tableau lors de sa déclaration par une liste de constantes de la façon suivante :

```
Type Tableau [10] = {C1, C2, ..., C10};
```

Par exemple, on peut écrire :

```
int Tab [4] = {1, 2, 3, 4};
```

Si le nombre de données dans la liste d'initialisation est inférieur à la dimension du tableau, seuls les premiers éléments seront initialisés.

Fiche N° 5
Les tableaux multidimensionnels

Il est également possible de déclarer un tableau à plusieurs dimensions. Par exemple, pour un tableau à deux dimensions :

```
type NomTableau [NLigs][NCols]
```

En fait, un tableau à deux dimensions est un tableau unidimensionnel dont chaque élément est lui-même un tableau.

On accède à un élément du tableau par l'expression

```
NomTableau [i][j].
```

Pour initialiser un tableau à plusieurs dimensions à sa déclaration, on utilise une liste dont chaque élément est une liste de constantes :

```
int Tab [2][3] = {{1, 2, 3}, {4, 5, 6}};
```

Fiche N° 6
Les chaînes de caractères

Une chaîne de caractères est une séquence de caractères textuels. En C, les chaînes de caractères sont représentées à l'aide de tableaux de caractères.

Par exemple,

```
char Prenom [20];
```

permet de représenter une chaîne d'au maximum 20 caractères.

Un tableau de caractères peut être initialisé par une liste de caractères comme dans

```
char Tab[7]={'e','x','e','m','p','l','e'};
```

mais aussi par une chaîne de caractères littérale :

```
char Tab[8]="exemple";
```

Dans ce cas d'initialisation, le compilateur complète la chaîne de caractères avec un caractère nul '\0'. Il faut donc que le tableau ait au moins un élément de plus que le nombre de caractères de la chaîne littérale.

Lors d'une initialisation, il est également possible de ne pas spécifier le nombre d'éléments du tableau.

```
char Tab []={'e','x','e','m','p','l','e'};
```

ou

```
char Tab []="exemple";
```

La taille de Tab sera égale au nombre de constantes de la liste d'initialisation (7) ou à la taille de la chaîne littérale (8).

Fiche N° 7
Structure de contrôle : Condition simple

Si le programmeur a besoin de spécifier qu'un seul parmi deux traitements différents doit être effectué, il doit utiliser la structure de contrôle conditionnelle.

Syntaxe :

```
if  (<expression logique>)
{
      <Suite d'instructions 1>
}
else
{
      <Suite d'instructions 2>
}
```

- Si <expression logique> est vraie, la suite d'instructions 1 sera exécutée sinon c'est la suite d'instructions 2 qui sera exécutée.
- La partie **"else"** est optionnelle. Elle peut être omise s'il n'y a pas d'instructions à effectuer lorsque la condition est fausse.
- Les accolades { } ne sont obligatoires que s'il y a plus d'une instruction dans le bloc du **if** ou du **else**.

Fiche N° 8
Structure de contrôle : Condition multiple

S'il y a plus de deux alternatives dans un programme, le langage C propose la structure **switch**. Dans cette structure, la valeur d'une expression déterminera le traitement à effectuer.

Syntaxe :

```
switch (<expression>)
{
  case <Valeur 1>:<Suite d'instructions 1>
  break;
  case <Valeur 2>:<Suite d'instructions 2>
  break;
  ...
  case <Valeur n>:<Suite d'instructions n>
  break;
  default : <Suite d'instructions n+1>
}
```

- L'expression utilisée dans **switch** doit être d'un type entier ou caractère.

Fiche N° 9
Structure de contrôle : itération

La structure itérative permet au programmeur de spécifier qu'un traitement (suite d'instructions) est à exécuter plusieurs fois. Appelée également boucle, l'itération admet 3 types :

- **La boucle "for"** : Le programmeur sait par avance combien de fois le traitement va se répéter.
- **La boucle "while"** : Le programmeur ne sait pas par avance combien de fois le traitement va se répéter mais il donne une condition qui, quand elle deviendra fausse, arrêtera les répétitions.
- **La boucle "do...while"** : Cette structure a la même logique que la boucle "while" sauf que la vérification de la condition se fait a posteriori. Ainsi le traitement se répètera au moins une fois.

Fiche N° 10
La boucle "for"

```
for (<Initialisation>;
     <Expression logique>;
     <Incrémentation>)
{
     <Suite d'instructions>
}
```

- L'instruction <Initialisation> est exécutée en premier. Elle sert généralement à initialiser l'indice de boucle.
- Ensuite <Expression logique> est évaluée. Si elle est vraie, l'expression <incrémentation> est évaluée puis <Suite d'instructions> est exécutée, sinon l'exécution de la boucle s'arrête.
- L'instruction <incrémentation> sert généralement à incrémenter l'indice de boucle.
- <Expression logique> est de nouveau évaluée, si elle est vraie la boucle continue, sinon elle s'arrête.

Fiche N° 11
Les boucles "while" et "do...while"

La boucle "while" :

```
while (<Expression logique>)
{
    <Suite d'instructions>
}
```

- <Expression logique> est évaluée. Si elle est vraie, <Suite d'instructions> est exécutée, puis <Expression logique> est évaluée à nouveau, etc.
- La suite d'instructions est exécutée 0 ou plusieurs fois.

La boucle "do...while" :

```
do
{
    <Suite d'instructions>
} while (<Expression logique>);
```

- Exécuter <Suite d'instructions> jusqu'à ce que <Expression logique> devienne fausse.
- <Suite d'instructions> est exécutée 1 ou plusieurs fois, <Expression logique> étant évaluée a posteriori.

Fiche N° 12
Les fonctions

Il est recommandé de diviser les programmes en plusieurs entités indépendantes plus petites. En C, ces entités sont appelées *fonctions*.

- **Améliorer la lisibilité**
  Un programme organisé en fonctions est plus lisible et plus compréhensible.
- **Faciliter la correction**
  Découvrir les erreurs et les corriger dans un ensemble de petites fonctions est plus facile que de le faire dans un programme long et complexe.
- **Faciliter la maintenance**
  Faire évoluer un programme structuré en fonctions est beaucoup plus aisé.
- **Favoriser la réutilisabilité**
  Les fonctions composant un programme peuvent être appelées plusieurs fois dans le même programme ou même à partir d'autres programmes.

- **Faciliter la conception**

  Le développement d'un programme est plus simple quand on le conçoit d'une manière organisée en fonctions.

- **Faciliter le travail de groupe**

  Un programme n'est pas forcément développé par une seule personne. Dans le cas de plusieurs développeurs, un programme segmenté en fonctions est plus facilement "partagé".

Fiche N° 13
Les paramètres

Les paramètres sont un moyen d'envoyer à une fonction des données et d'en récupérer des résultats. Ce sont des constantes ou des noms de variables mises entre parenthèses après le nom de la fonction dans l'instruction d'appel.

Les paramètres sont de 2 types :

- **les paramètres "données"** : Ce sont des constantes ou des noms de variables ayant reçu des valeurs avant l'appel. Ils permettent d'envoyer des données à la fonction appelée.
- **les paramètres "résultats"** : Une fonction peut calculer ou déterminer des valeurs et les transmettre comme résultats de son traitement. Une variable utilisée comme paramètre "résultat" peut en même temps être utilisée comme paramètre "donnée" si elle est initialisée avant l'appel de la fonction et si sa valeur initiale est utilisée à l'intérieur de la fonction.

Les paramètres "résultats" doivent être envoyés par adresse (on dit également par référence). Les paramètres "données" sont envoyés par valeur.

Exemples :

Afficher (X); permet d'afficher la valeur d'un entier donné. X est le paramètre "donnée" de la fonction.

Saisir (&Y); permet de saisir un entier à partir du clavier. Y est le paramètre "résultat" de cette fonction.

Perm (&X, &Y) ; X et Y sont des paramètres "données" et "résultats" de cette fonction. Ils ont des valeurs avant l'appel de la fonction. Ils auront d'autres valeurs après l'appel. La fonction permute les contenus de X et Y.

Fiche N° 14
La récursivité

D'une manière générale, une définition récursive est une définition qui utilise le mot ou le concept qu'elle définit. L'exemple classique d'une définition récursive est celui de la factorielle:

$$0! = 1 ; \qquad n! = n * (n-1)! \quad n > 0$$

En programmation, une fonction récursive est une fonction qui fait appel à elle-même. Il faut toutefois qu'elle comporte une partie conditionnée non récursive, sinon les appels récursifs ne s'arrêtent jamais. Exemple:

```
Recursion (Parametres)
{
  if (TEST)
  {
    <Suite d'instructions>
    // Instructions du point d'arrêt
  }
  else
  {
    <Suite d'instructions>
    Recursion (Nouveaux Parametres);
  }
}
```

ÉNONCÉS

Série N° 1

Points abordés

- Variables
- Affectation

Exercice 1.1 : Noms valides de variables

Dites si ces noms de variables sont valides ou pas:
NOMBRE, ValeurMaximale, x, mon_age, x1, 1x, max%, valeur, mon-age, mon age, procedure, km/hr, is, mon_âge, Nombre d'éléments, case.

Exercice 1.2 : Choix des noms de variables

Donner un nom de variable approprié pour les informations suivantes:

- Le nombre d'étudiants par groupe
- Le nombre de groupes par filière
- Le nombre de buts marqués dans un match de football
- Le numéro d'une carte d'identité nationale
- L'âge d'un étudiant

Exercice 1.3 : Affectations valides

Dites si ces instructions sont correctes ou pas:

- `A1 = 3;`
- `B = 3 * 5;`
- `C = A1;`
- `A + 1 = B + 1; /`
- `C = (B + 5) * 4;`
- `B = B + C;`
- `A = B x C;/`
- `A  = 3C; /`
- `A := B * C;/`

Exercice 1.4 : Résultats d'affectations

Donner le contenu des variables X, Y et Z après l'exécution de chacune des instructions suivantes :

1)
```
X = 5;
Y = 3;
Z = X + Y;
X = 2;
Z = Y - X;
```

2)
```
X = 5;
Y= X + 4;
X = X + 1;
Y = X - 4;
Z = 5;
```

3)
```
X = 2;
Y = 3;
X = X + Y;
Y = X - Y;
X = X - Y;
```

4)
```
X = 5;
Y = 10;
Z = X + Y;
Y = X + Y;
X = Z;
```

5)
```
X = 5;
Y = 7;
X = Y;
Y = X;
```

Série N° 2

Points abordés

- Affectation
- Affichage
- Saisie
- Expressions

Exercice 2.1 : Saisie, calculs, affichages

Écrire un programme qui permet de lire au clavier un entier et d'afficher à l'écran les valeurs résultant de sa multiplication par 2, 3, 5, 7 et 9.

Exercice 2.2 : Somme de deux entiers

Écrire un programme qui permet de calculer la somme de deux entiers donnés et d'afficher le résultat de cette somme.

Exercice 2.3 : Permutation de deux entiers

Écrire un programme qui permet de permuter le contenu de deux variables contenant deux entiers donnés.

Exercice 2.4 : Périmètre et surface d'un rectangle

Écrire un programme qui permet de calculer le périmètre et la surface d'un rectangle dont les dimensions sont données.

Exercice 2.5 : Quotient et reste d'une division entière

Écrire un programme qui calcule le quotient et le reste de la division entière de deux nombres positifs A et B sachant que B est supposé non nul.

Exercice 2.6 : Conversion horaire

Écrire un programme qui lit un nombre positif de secondes et le convertit en heures, minutes et secondes.

Série N° 3

Points abordés

- Condition simple
- Condition multiple
- Schémas de contrôle conditionnels

Exercice 3.1 : Maximum de deux entiers (puis de trois)

a) Écrire un programme qui permet d'afficher le maximum de deux entiers donnés.

b) Écrire un programme qui permet d'afficher le maximum de trois entiers donnés.

Exercice 3.2 : Ordre croissant de trois entiers

Écrire un programme qui permet d'afficher dans l'ordre croissant trois entiers donnés.

Exercice 3.3 : Affranchissement d'une lettre

Les tarifs d'affranchissement d'une lettre selon le poids sont les suivants :

- En dessous de 20 grammes : 0,250 Dinar.
- À partir de 20 grammes, mais au dessous de 50 grammes : 0,500 Dinar.
- À partir de 50 grammes : 1 Dinar.

Écrire un programme permettant de calculer le tarif pour un poids donné.

Exercice 3.4 : Mention d'un étudiant

Écrire une programme qui permet de lire la moyenne d'un étudiant et d'afficher la mention correspondante (Note $\geq$ 16 : Très bien; 14 $\leq$ Note < 16 : Bien; 12 $\leq$ Note < 14 : Assez bien; 10 $\leq$ Note < 12 : Passable; 0 $\leq$ Note < 10 : Echec).

Exercice 3.5 : Équation du premier degré

Écrire un programme permettant de résoudre une équation du premier degré à une seule inconnue (ax + b = 0). Les entrées sont les coefficients a et b.

Exercice 3.6 : Équation du second degré

Écrire un programme qui permet de résoudre une équation du second degré à une seule inconnue ($ax^2 + bx + c = 0$). Les entrées sont les coefficients a, b et c.

Exercice 3.7 : Salaire hebdomadaire d'un employé

Une entreprise paie ses salariés selon le principe d'un taux horaire de base auquel on applique un coefficient K de la manière suivante :

- Pour 39 heures de travail par semaine, K = 1.
- De la $40^{ème}$ à la $44^{ème}$ heure par semaine, K = 1,5.
- Après la $44^{ème}$ heure, K = 2.

Écrire un programme qui permet de calculer le salaire hebdomadaire d'un employé sachant son nombre d'heures travaillées et le taux horaire de base.

Exercice 3.8 : Montant des impôts

Écrire un programme permettant de calculer le montant des impôts dus par un contribuable en fonction de son revenu imposable (Rimp) et son nombre de parts fiscales (Nparts) qui devront être saisis au clavier. Les règles suivantes doivent être respectées :

1. Le revenu par part (Rpart) est égal au quotient Rimp par Nparts.
2. L'impôt par part (Impart) est calculé selon le barème suivant :
   - 0 si Rpart est inférieur à 50 Dinars.
   - 10% sur la tranche de Rpart comprise entre 50 Dinars et 100 Dinars.
   - 25% sur la tranche de Rpart comprise entre 100 Dinars et 200 Dinars.
   - 50% sur la tranche de Rpart qui dépasse 200 Dinars.
3. L'impôt total est égal au produit de NbParts par Impart.

Exercice 3.9 : Calculatrice

Écrire un programme qui permet d'effectuer les opérations simples (+, -, *, /). A l'exécution, le programme donnerait par exemple :

5 + 11 = 16

Exercice 3.10 : Date du lendemain

Écrire un programme qui permet de déterminer la date du lendemain à partir d'une date du jour représentée par 3 valeurs entières (jour, mois et année).

Série N° 4

Points abordés

- Itération
- Boucle "for"
- Boucle "while"
- Schémas de contrôle itératifs

Exercice 4.1 : Somme des inverses

Écrire un programme qui permet de calculer la somme suivante pour un entier N donné :

$$S = \sum_{i=1}^{N} 1/i$$

Exercice 4.2 : Lettres de l'alphabet

Écrire un programme qui affiche toutes les lettres de l'alphabet latin (minuscules et majuscules).

Exercice 4.3 : Table de multiplication

Écrire un programme qui affiche les résultats de la multiplication d'un entier donné par les valeurs de 1 à 10. Pour la valeur exemple 5, l'affichage résultat aura l'aspect suivant :

```
1   2   3   4   5   6   7   8   9   10
-------------------------------------------------------------
5   10  15  20  25  30  35  40  45  50
```

Exercice 4.4 : Table de multiplication symétrique

Écrire un programme qui affiche la table de multiplication jusqu'à un entier donné. Pour la valeur 5, l'affichage résultat aura par exemple l'aspect suivant :

	1	2	3	4	5
1	1	2	3	4	5
2	2	4	6	8	10
3	3	6	9	12	15
4	4	8	12	16	20
5	5	10	15	20	25

Exercice 4.5 : Somme, moyenne, ...

Écrire un programme qui permet de calculer la somme, la moyenne, la valeur la plus grande, la valeur la plus petite et le nombre de valeurs positives d'un ensemble d'entiers saisis. Le nombre d'entiers à saisir est une donnée.

Exercice 4.6 : Problème de démographie

En juillet 1998, la Tunisie comptait 9 380 404 habitants (32% des personnes ont un âge allant de 0 à 14 ans). Les taux de naissance et de décès étaient respectivement de 20.07‰ et 5.06‰. Le taux net de migration est de -0.73‰.

En fonction de ces données et en supposant ces taux constants, écrire un programme qui permet de calculer le nombre d'habitants de la Tunisie au début d'une année donnée. L'année donnée doit être postérieure à 1998.

Exercice 4.7 : Travail dangereux

On vous propose un travail dangereux. Le salaire quotidien commence à 0,001 Dinar est multiplié par 2 tous les jours. Ainsi, vous recevrez 0,001 Dinar le premier jour, 0,002 Dinar le deuxième jour, 0,004 Dinar le troisième jour, et ainsi de suite. Vous voulez savoir combien de jours faut-il travailler pour devenir millionnaire en dinars. Écrire un programme qui permet de le déterminer.

Exercice 4.8 : Multiplication russe

La multiplication russe est une méthode particulière permettant de multiplier deux entiers strictement positifs en utilisant la division entière par 2, la multiplication par 2 et l'addition.
Voici comment on procède. Soient A et B deux entiers strictement positifs. La méthode consiste à mettre A dans une première colonne et B dans une deuxième colonne. Il s'agit ensuite de répéter les deux opérations suivantes :

- Division de la dernière valeur de la colonne A par 2
- Multiplication de la dernière valeur de la colonne B par 2

et ce jusqu'à atteindre la valeur 1 dans la première colonne. Le résultat de la multiplication de A par B s'obtient en additionnant les nombres de la 2ème colonne qui sont en face de nombres impairs dans la 1ère colonne. Les nombres de la seconde colonne qui sont en face de nombres pairs seront ignorés.

Exemple 1		**Exemple 2**	
A	**B**	**A**	**B**
9	**11**	12	5 (ignoré car 12 est pair)
4	22 (ignoré car 4 est pair)	6	10 (ignoré car 6 est pair)
2	44 (ignoré car 2 est pair)	3	**20**
1	**88**	1	**40**
9 * 11 = 11+88 =99		12 * 5 = 20 + 40 = 60	

Écrire un programme permettant de multiplier deux entiers strictement positifs A et B en utilisant la méthode de la multiplication russe.

Série N° 5

Points abordés

- Tableaux à une dimension
- Itérations

Exercice 5.1 : Permutation de deux cases d'un tableau

Écrire un programme qui permet de permuter les valeurs de deux cases dans un tableau d'entiers.

Exercice 5.2 : Nombre de valeurs paires et impaires

Écrire un programme qui permet de saisir un nombre variable d'entiers pour ensuite compter parmi ces entiers le nombre de valeurs paires et le nombre de valeurs impaires.

Exercice 5.3 : Produits des éléments d'un tableau

1) Écrire un programme qui permet de donner le produit des éléments d'un tableau de réels.
2) Écrire un programme qui permet de donner le produit des éléments non nuls d'un tableau de réels.
3) Modifier ce dernier programme pour avoir également la valeur moyenne des éléments du tableau.

Exercice 5.4 : Indice de la plus grande valeur d'un tableau

Écrire un programme qui, dans un tableau d'entiers donné, recherche l'indice de la plus grande valeur et celui de la plus petite valeur (la première occurrence apparue).

Exercice 5.5 : Recherche d'une valeur dans un tableau

Écrire un programme qui recherche dans un tableau d'entiers l'indice d'une valeur donnée (la première occurrence apparue).

Exercice 5.6 : Recherche dans un tableau d'entiers ordonnés

Écrire un programme qui recherche dans un tableau d'entiers ordonnés dans le sens croissant l'indice d'une valeur donnée.

Exercice 5.7 : Permutation circulaire

Écrire un programme qui effectue une permutation circulaire des valeurs d'un tableau d'entiers. Chaque case doit prendre la valeur de sa suivante. La dernière case doit prendre la valeur de la première.

Exercice 5.8 : Comparaison de deux tableaux

Écrire un programme qui compare deux tableaux d'entiers pour dire s'ils sont identiques ou non. Deux tableaux sont identiques s'ils ont la même taille et qu'ils contiennent les mêmes valeurs dans le même ordre.

Exercice 5.9 : Inversion d'un tableau

Écrire un programme qui permet d'inverser un tableau d'entiers.

Exercice 5.10 : Insertion dans un tableau

Écrire un programme qui insère une valeur donnée v dans un tableau d'entiers T donné à un indice donné i. Les cases suivant l'indice i doivent être "décalées" et la valeur donnée v sera mise dans la case n°i.

Exercice 5.11 : Suppression d'un tableau

Écrire un programme qui supprime une case dont l'indice I est donné d'un tableau d'entiers T. Les cases suivant l'indice I doivent être "décalées".

Exercice 5.12 : Séquence ordonnée la plus longue

Écrire un programme qui permet de calculer la taille de la séquence d'entiers ordonnés (d'une manière croissante) la plus longue dans un tableau d'entiers.

Exercice 5.13 : Affichage sans répétitions

Écrire un programme qui permet de saisir un ensemble d'entiers puis de les afficher sans répétitions.

Série N° 6

Points abordés

- Tableaux de caractères
- Itérations

Exercice 6.1 : Formats de date

Une date est représentée sous la forme d'une chaîne de 8 caractères. Par exemple "22/02/99".

1) Écrire un programme qui permet, à partir d'une date au format français "JJ/MM/AA", de calculer la même date au format américain "MM/JJ/AA".

2) Écrire un programme qui permet, à partir d'une date au format français "JJ/MM/AA", de calculer la même date au format ISO "AAAA/MM/JJ".

Exercice 6.2 : Nombre d'occurrences d'un caractère

Écrire un programme qui compte les occurrences d'un caractère dans un tableau de caractères.

Exercice 6.3 : Nombre d'occurrences d'un bigramme

On appelle bigramme une suite de deux lettres. Écrire un programme qui calcule le nombre d'occurrences d'un bigramme donné dans un tableau de caractères.

Exercice 6.4 : Occurrences de toutes les lettres de l'alphabet

Écrire un programme qui permet de calculer les nombres d'occurrences de toutes les lettres minuscules de l'alphabet dans une chaîne de caractères.

Exercice 6.5 : Recherche d'une sous-chaîne dans une chaîne

Écrire un programme qui recherche une sous-chaîne de caractères dans une chaîne de caractères. Par exemple, si la chaîne de caractères contient la valeur "programme d'ordinateur" et que la sous-chaîne

recherchée est "gramme d'or", la procédure de recherche doit indiquer que la sous-chaîne existe et l'indice retourné doit être égal à 3.

Exercice 6.6 : Palindrome

Un mot est dit Palindrome s'il se lit de la même manière dans le sens de lecture de droite à gauche que dans le sens de gauche à droite. Par exemple, les mots "LAVAL" et "ADA" sont des palindromes. Écrire un programme qui dit si une chaîne donnée est un palindrome.

Exercice 6.7 : Éliminer les espaces superflus

On veut nettoyer un tableau de caractères en éliminant les espaces superflus : chaque fois que l'on trouve une suite d'espaces, il faudra les éliminer tous sauf un. Sachant que le premier caractère du tableau et le dernier caractère sont différents du caractère espace.

Série N° 7

Points abordés

- Tableaux à deux dimensions
- Matrices
- Itérations

Exercice 7.1 : Matrice symétrique

Écrire un programme qui permet de déterminer si une matrice n x n est symétrique. Une matrice symétrique a la propriété suivante : $\forall i$, $\forall j$, aij = aji.

Exercice 7.2 : Multiplication de deux matrices

Écrire un programme qui permet d'effectuer la multiplication de deux matrices A et B avec respectivement les dimensions m x n et n x p. Noter que le nombre de colonnes de A doit être égal au nombre de lignes de B. Le résultat de la multiplication est une matrice C de dimensions m x p avec

$$c_{ij} = \sum_{k=1}^{n} a_{ik} \times b_{kj}$$

$$\begin{vmatrix} 1 & 2 \\ 4 & 7 \\ 8 & 3 \end{vmatrix} \quad X \quad \begin{vmatrix} 1 & 2 & 0 & 2 \\ 3 & 4 & 3 & 1 \end{vmatrix} \quad = \quad \begin{vmatrix} 7 & 10 & 6 & 4 \\ 25 & 36 & 21 & 15 \\ 17 & 28 & 9 & 19 \end{vmatrix}$$

Exercice 7.3 : Extraction d'une ligne ou d'une colonne

Écrire un programme qui permet d'extraire une ligne ou une colonne dont l'indice est donné à partir d'un tableau à deux dimensions.

Exercice 7.4 : Carré magique

Un carré magique est un arrangement de nombres avec n lignes et n colonnes. La somme des valeurs dans chaque ligne, colonne et diagonale est la même. Par exemple, le carré suivant est magique.

16	9	2	7
6	3	12	13
11	14	5	4
1	8	15	10

1) Écrire un programme qui permet de déterminer si un carré est magique.
2) Une autre condition pour un arrangement de nombres avec n lignes et n colonnes d'être un vrai carré magique est qu'il doit contenir tous les entiers 1, 2, ..., n^2. Améliorer votre programme pour qu'il vérifie également cette condition.

Série N° 8

Points abordés

- Fonctions
- Modules
- Sous-programmes
- Itérations

Exercice 8.1 : Maximum de trois nombres

Écrire une fonction qui retourne le maximum de trois nombres entiers donnés, en utilisant la fonction qui donne le maximum de deux nombres.

Exercice 8.2 : Somme des éléments d'un tableau

Écrire une fonction qui calcule la somme des éléments d'un tableau d'entiers.

Exercice 8.3 : Compter le nombre de voyelles

Écrire une fonction qui permet de compter le nombre de voyelles (minuscules) d'une chaîne de caractères donnée.

Exercice 8.4 : Un entier dans un tableau à 2 dimensions

Écrire une fonction qui permet de rechercher un entier donné dans un tableau d'entiers à deux dimensions.

Exercice 8.5 : Produit d'entiers d'un tableau à 2 dimensions

Écrire une fonction qui permet de calculer le produit de tous les entiers d'un tableau à 2 dimensions. Le tableau et ses dimensions sont passés à la fonction en paramètres.

Exercice 8.6 : Transposition d'une matrice

Écrire une fonction permettant de transposer une matrice (tableau à double dimension). Il ne faut pas utiliser un autre tableau. Le traitement se fera dans la même variable.

Exemple :

1	2	3	==>	1	4	7
4	5	6		2	5	8
7	8	9		3	6	9

Exercice 8.7 : Insertion d'un tableau d'entiers dans un autre

Écrire une fonction qui permet d'insérer un tableau d'entiers *T2* dans un autre tableau d'entiers *T1* à un indice donné *Indice*. *T1* et *T2* contiennent respectivement *N1* et *N2* éléments.

Exemple :

T1

1	*4*	*5*	*8*	*9*					

T2

2	*3*	*6*							

Indice 2

⇨ **T1**

1	4	2	3	6	5	8	9		

Exercice 8.8 : Suppression d'entiers d'un tableau

Écrire une fonction qui permet de supprimer N entiers d'un tableau d'entiers *T1* à partir d'un indice donné *Indice*.

Exemple :

T1

3	*5*	*9*	*11*	*13*					

N *2*

Indice 1

⇨ **T1**

3	11	13							

Exercice 8.9 : Sens du tri d'un tableau

Écrire une fonction (TriéComment) dont le résultat est un entier pouvant avoir les valeurs suivantes :

- 1 : si le tableau est trié dans le sens croissant.
- 2 : si le tableau est trié dans le sens décroissant.
- 3 : si le tableau n'est pas trié.

Cette fonction ne doit parcourir le tableau qu'une seule et unique fois.

Série N° 9

Points abordés

- Récursivité

Exercice 9.1 : Factorielle d'un entier

Écrire une fonction récursive qui permet de calculer la factorielle d'un entier positif.

Exercice 9.2 : Somme des premiers entiers

Écrire une fonction récursive qui permet de calculer la somme des n ($n \geq 0$) premiers termes (à partir de 1 jusqu'à n).

Exercice 9.3 : Nombre de Fibonacci

Écrire une fonction récursive qui permet de calculer les nombres de "Fibonacci" qui sont définis par :

Fibonacci (0) = 0,
Fibonacci (1) = 1,
et pour tout n > 1, Fibonacci (n) = Fibonacci (n-1) + Fibonacci (n-2)

Exercice 9.4 : PGCD de deux entiers

Écrire une fonction récursive qui permet de calculer le plus grand diviseur commun de deux entiers naturels strictement positifs.

Exercice 9.5 : Fonction d'Ackermann

Écrire une fonction récursive qui permet de calculer la valeur de la fonction d'Ackermann définie de la manière suivante pour $m \geq 0$ et $n \geq 0$:

Acker(m, n) = Acker(m-1, Acker(m, n-1))
Acker(0, n) = n+1 pour n > 0
Acker(m, 0) = Acker(m-1, 1) pour m > 0

Exercice 9.6 : Somme des entiers de même parité

Il s'agit de calculer la somme des entiers allant de 0 à un entier donné N et qui ont la même parité que N. Par exemple,
Pour la valeur 6, la fonction doit rendre la valeur 12. (6+4+2+0 = 12)
Pour la valeur 5, la fonction doit rendre la valeur 9. (5+3+1 = 9)

a) Écrire une fonction itérative permettant de résoudre ce problème.
b) Résoudre le même problème par une fonction récursive.

Exercice 9.7 : Occurrences d'un entier dans un tableau

Écrire une fonction récursive qui compte le nombre d'occurrences d'un entier dans un tableau d'entiers.

Exercice 9.8 : Inversion d'un tableau

Écrire une fonction récursive qui permet d'inverser un tableau d'entiers.

Exercice 9.9 : Palindrome

Écrire une fonction récursive qui permet de dire si un mot donné est un palindrome ou pas.

Exercice 9.10 : Résultat d'exécution de Guess

Donner le résultat de : `Guess (3, 1);` si la fonction *Guess* a la définition suivante :

```
int Guess (int x, int y)
{
 if (x == y)
  return x;
 else
 {
  if (x > y)
   return Guess (x-1, y) + Guess (x, y+1);
  else
   return Guess (x+1, y) + Guess (x, y-1);
 }
}
```

Série N° 10

Points abordés

- Problèmes de synthèse

Exercice 10.1 : Conversion en majuscules

a) Écrire une fonction (*MinMaj*) qui reçoit un caractère et dont le résultat est un caractère. Si le caractère donné est une lettre minuscule la fonction rend la même lettre en majuscule. Si le caractère donné est une lettre majuscule, la fonction rend la même lettre en minuscule. Dans les autres cas, la fonction rend le même caractère reçu en entrée.

b) Écrire une fonction (*MinStr*) qui transforme les lettres majuscules d'une chaîne de caractères en lettres minuscules.

Exercice 10.2 : Nombres premiers

Écrire un programme permettant d'afficher l'ensemble des nombres premiers entre 2 et n (n est un entier supérieur ou égal à 2). Exemple : Pour N = 12, la liste des nombres premiers est : 2, 3, 5, 7 et 11.

Exercice 10.3 : Nombres premiers dans une matrice

Écrire une fonction permettant de compter le nombre de nombres premiers se trouvant dans une matrice.

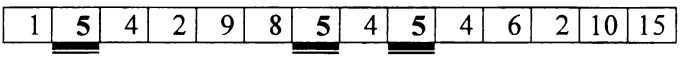
Exercice 10.4 : Recherche double

Écrire une fonction qui reçoit deux entiers A et B et un tableau d'entiers T et qui permet de rechercher A et B dans T. Le résultat de cette fonction doit être 0, 1 OU 2 respectivement si aucune des deux valeurs A et B n'existe dans T, si l'une seulement des valeurs A et B existe dans T OU si les deux valeurs A et B existent dans T. Dans cette fonction, le tableau T ne doit être parcouru qu'une seule et unique fois.

Exercice 10.5 : Valeur la plus fréquente

Écrire une fonction qui permet de trouver la valeur la plus fréquente dans un tableau d'entiers qu'elle reçoit comme paramètre. S'il y a ambiguïté, donner la valeur qui apparaît la première dans le tableau.

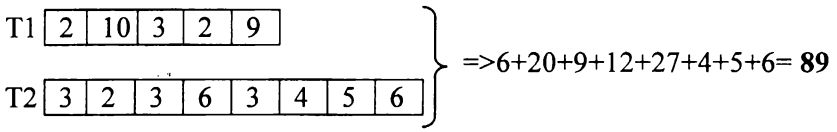
Exemple : Pour le tableau :



Le résultat de la fonction devrait être : 5

Exercice 10.6: Somme des produits des valeurs de 2 tableaux

Écrire une fonction en C qui reçoit comme paramètres deux tableaux d'entiers et leurs tailles. Cette fonction calcule la **somme des produits** des valeurs des deux tableaux. Dans le cas où un tableau est plus grand que l'autre, les valeurs du plus grand tableau qui n'ont pas de correspondant dans le plus petit tableau seront multipliées par 1.



Exercice 10.7 : Recherche d'une somme

Écrire une fonction qui reçoit en paramètres un tableau d'entiers T et une valeur entière V. Cette fonction permet de trouver s'il existe deux cases dans T qui ont une somme égale à V. Si oui, elle rend les indices des deux premières cases qui satisfont au critère demandé.

Exercice 10.8 : Paire dont la somme est maximale

Écrire une fonction qui permet de trouver la paire de valeurs consécutives ayant la somme maximale dans un tableau d'entiers qu'elle reçoit comme paramètre. Le résultat sera l'indice de la première valeur de la paire ainsi que la somme calculée. Exemple :

Pour le tableau :

1	5	4	2	9	8	6	2	10	1

Le résultat sera I = 5 et S = 17

Exercice 10.9 : Tailles des séquences ordonnées

Écrire une fonction qui permet de calculer pour un tableau d'entiers donné la taille de la séquence ordonnée dans le sens croissant la plus longue et la taille de la séquence ordonnée dans le sens décroissant la plus longue. Cette fonction ne doit effectuer qu'un seul et unique parcours du tableau. Exemple: pour le tableau **4 5 6 7** 6 8 2 **9 2 1** 2 5 6 on aurait les résultats 4 et 3.

Exercice 10.10 : Taille de la séquence désordonnée

Appelons une séquence désordonnée toute suite d'éléments dans un tableau dans laquelle il n'existe pas trois éléments triés (ni dans le sens croissant, ni décroissant). Exemple : 4 5 **6 7 6 8 2 9 1 2 0 5** 6. Écrire une fonction qui permet de calculer pour un tableau d'entiers donné la taille de la séquence désordonnée la plus longue.

Exercice 10.11 : Substitution de valeurs

1. Écrire une fonction qui reçoit deux tableaux d'entiers (*TIN* et S2) et un entier (E). Cette procédure rend un tableau d'entiers (*TOUT).* *TOUT* contiendra toutes les valeurs de TIN sauf que les cases contenant la valeur E seront remplacées par la suite de cases de S2.

Tableau en entrée TIN :

15	3	4	5	9	8	3	4	3	4	5	9	10	15

E (donnée) : 3 S2 (donnée) :

5	9

Tableau résultat TOUT :

15	5	9	4	5	9	8	5	9	4	5	9	4	5	9	10	15

2. Écrire une fonction qui a les mêmes paramètres que P1 sauf qu'ici *TOUT* contiendra toutes les valeurs de TIN où les suites de cases S2 ont été remplacées par la valeur E.

Tableau en entrée TIN

15	3	4	5	9	8	3	4	3	4	5	9	10	15

E (donnée) : 3

S2 (donnée) :

3	4	5

Tableau résultat TOUT

15	3	9	8	3	4	3	9	10	15

Exercice 10.12 : Fusion de deux tableaux

1) Écrire une fonction qui, à partir de deux tableaux dont les cases sont triées dans l'ordre croissant, construit un troisième tableau contenant la fusion de deux tableaux en entrée. Ce tableau résultat doit également être ordonné dans le sens croissant.

Exemple :

T1	1	4	5	8	9			
T2	2	3	6					
⇨ T3	1	2	3	4	5	6	8	9

2) Résoudre le même problème en produisant un tableau résultat ordonné dans le sens décroissant.

Exercice 10.13 : Anagrammes

Deux mots sont dits *Anagrammes* s'ils contiennent exactement les mêmes lettres. Par exemple les mots {"carte", "trace"}, {"viral",

"rival", "avril", "livra"} et {"crier", "cirer"} sont des ensembles d'anagrammes. Écrire une fonction qui détermine si deux mots sont anagrammes.

Exercice 10.14 : Tableaux équivalents

Écrire une fonction (*Equivalents*) qui compare deux tableaux d'entiers pour dire s'ils sont équivalents ou non. Deux tableaux sont équivalents s'ils ont le même nombre d'éléments et qu'ils contiennent les mêmes valeurs, l'ordre des valeurs n'ayant pas d'importance.

Exercice 10.15 : Comparaison de situations de tri

Écrire une fonction (*MemeOrdre*) qui détermine si deux tableaux d'entiers sont dans la même situation de tri. Le résultat de cette fonction est une valeur logique. La valeur logique (VRAI) est retournée si et seulement si l'un de ces trois cas se présente :

- Les deux tableaux sont triés dans le sens croissant
- Les deux tableaux sont triés dans le sens décroissant
- Les deux tableaux ne sont pas triés

Les deux tableaux peuvent être de tailles différentes. Cette fonction ne doit parcourir les deux tableaux qu'une seule et unique fois. De préférence, cette fonction ne comportera qu'une seule boucle.

<u>Exemple 1 :</u>

T1: | 6 | 5 | 4 | 3 | 2 | 1 |

⇨ VRAI

T2: | 8 | 7 | 6 | 5 | 5 | 4 |

<u>Exemple 2 :</u>

T1: | 1 | 2 | 3 | 4 | 5 | 6 |

⇨ VRAI

T2: | 1 | 2 | 2 | 5 | 7 | 9 | 9 | 10 | 11 | 15 |

<u>Exemple 3 :</u>

T1: | 1 | 2 | 3 | 2 | 1 | 4 |

⇨ VRAI

T2: | 5 | 3 | 6 | 4 | 7 |

Exercice 10.16: Nombre de mots en commun

Écrire un programme C qui permet de :

1. Saisir deux chaînes de caractères représentant deux phrases.
2. Compter le nombre de mots en commun entre les deux phrases.
3. Afficher le résultat.

Un mot est une suite de caractères alphanumériques délimitée par des séparateurs appartenant à la liste exclusive suivante :

' : L'apostrophe -, : La virgule -;: Le point virgule - . : Le point - : L'espace.

Il est à remarquer que les lettres majuscules et minuscules ne sont pas équivalentes (Mot != mot) et que les séparateurs ne sont pas comptés comme étant des mots.

Exemple 1:

Pour les deux chaînes :

Ch1: *Il est important de partir. C'est une question de bon sens.*

Ch2: *Ce n'est pas partir qui est important. Trouver un sens compte plus.*

Le programme devra afficher : *Le nombre de mots en commun est 5.*

Exemple 2:

Pour les deux chaînes :

Ch1: *Pour être un bon programmeur, il faut programmer vite et bien.*

Ch2: *Pour qu'un programme soit correct, il faut qu'il donne le bon résultat.*

Le programme devra afficher : *Le nombre de mots en commun est 5.*

Il est à remarquer que les mots qui se répètent dans la même phrase se comptent autant de fois qu'ils se repètent.

Série N° 11

Points abordés

- Tris de tableaux

Exercice 11.1 : Tri par sélection

Écrire une fonction qui fait le tri par sélection.

Exercice 11.2 : Tri à bulles

Écrire une fonction qui fait le tri à Bulles.

Exercice 11.3 : Tri par insertion

Écrire une fonction qui fait le tri par insertion.

Exercice 11.4 : Tri rapide

Écrire une fonction qui fait le tri rapide.

CORRIGÉS

Série N° 1

Exercice 1.1 : Noms valides de variables

Nom de variable	Valide	Non valide
NOMBRE	✓	
ValeurMaximale	✓	
x	✓	
mon_age	✓	
x1	✓	
1x		✖ (il commence par un chiffre)
max%		✖ (il contient %)
valeur	✓	
mon-age		✖ (il contient le caractère -)
mon age		✖ (il contient le caractère espace)
Procedure	✓	
km/hr		✖ (il contient le caractère /)
Is	✓	
mon_âge		✖ (il contient le caractère â)
Nombre d'éléments		✖ (il contient le caractère espace, les accents et l'apostrophe)
Case		✖ (c'est un mot-clé du langage C)

1. *Le nom d'une variable doit impérativement commencer par une lettre et ne peut contenir que des lettres, des chiffres et le caractère de soulignement '_'.*

2. *Attention : Les variables 'Nombre' et 'nombre' sont différentes. Le langage C est sensible au cas. Les lettres majuscules et minuscules sont différentes.*

Exercice 1.2 : Choix des noms de variables

- Nbre_ Etud_Grp, NbEtudGrp
- Nbre_Grp_Fil, NbrGrpFil
- Nbre_But_Match_F, NbrButMatch
- NCIN
- Age_Etud, AgeEtudiant

Donner un nom à une variable dépend toujours du contexte du programme. Il faut penser à choisir des noms significatifs et non ambigus.

Exercice 1.3 : Affectations valides

Instruction	Valide	Non valide
A1 = 3;	✓	
B = 3 * 5;	✓	
C = A1;	✓	
A + 1 = B + 1;		✗ (L'affectation doit se faire dans une variable et non dans une expression « A +1 »)
C = (B + 5) * 4;	✓	
B = B + C;	✓	
B = B x C;		✗ (Le symbole de la multiplication en C est '*')
A = 3C		✗ (3C ne peut pas être une variable. L'expression serait plutôt : 3 * C)
A := B * C ;		✗ (L'affectation en C s'effectue au moyen du symbole '=' et non ' :=')

Exercice 1.4 : Résultats d'affectations

1)

Instruction	X	Y	Z
X = 5;	5	-	-
Y = 3;	5	3	-
Z = X + Y;	5	3	8
X = 2;	2	3	8
Z = Y – X;	2	3	1

2)

Instruction	X	Y	Z
X = 5;	5	-	-
Y = X + 4;	5	9	-
X = X + 1;	6	9	-
Y = X - 4;	6	2	-
Z = 5;	6	2	5

3)

Instruction	X	Y
X = 2;	2	-
Y = 3;	2	3
X = X + Y;	5	3
Y = X – Y;	5	2
X = X – Y;	3	2

4)

Instruction	X	Y	Z
X = 5;	5	-	-
Y = 10;	5	10	-
Z = X + Y;	5	10	15
Y = X + Y;	5	15	15
X = Z;	15	15	15

5)

Instruction	X	Y
X = 5;	5	-
Y = 7;	5	7
X = Y;	7	7
Y = X;	7	7

Série N° 2

Exercice 2.1 : Saisie, calculs, affichages

```
#include <stdio.h>
int main ()
{
   int x;
   // Lecture de l'entier à partir du clavier
   printf ("Donnez un entier : ");
   scanf ("%d", &x);

   // Affichage du résultat
   printf ("\n %d x 2 = %d ", x, x * 2);
   printf ("\n %d x 3 = %d ", x, x * 3);
   printf ("\n %d x 5 = %d ", x, x * 5);
   printf ("\n %d x 7 = %d ", x, x * 7);
   printf ("\n %d x 9 = %d ", x, x * 9);

   return 0 ;
}
```

Exercice 2.2 : Somme de deux entiers

```
#include <stdio.h>
int main ()
{
   int x, y, s;

   // Saisie des deux entiers à additionner
   printf ("1er entier : "); scanf ("%d", &x);
   printf ("2ème entier: "); scanf ("%d", &y);
```

```
   // Traitement
   s = x + y;

   // Affichage du résultat
   printf ("%d + %d = %d", x, y, s);

   return 0 ;
}
```

On pourrait se passer de la variable s en affichant directement x+y comme suit :

```
printf ("%d + %d = %d", x, y, x+y);
```

Exercice 2.3 : Permutation de deux entiers

Première méthode : Utilisation d'une variable intermédiaire

```
#include <stdio.h>
int main()
{
   int x, y, aux;

   // Saisie des deux entiers à permuter
   printf ("1er  entier : "); scanf ("%d", &x);
   printf ("2ème entier : "); scanf ("%d", &y);

   // Traitement
   aux = x;
   x   = y;
   y   = aux;

   // Affichage des résultats
   printf ("1er entier  = %d \n", x);
   printf ("2ème entier = %d \n", y);
   return 0 ;
}
```

Quand l'énoncé demande d'écrire un programme "qui calcule", le programme doit calculer et... afficher ses résultats.

Deuxième méthode : Sans recours à une variable intermédiaire

```
#include <stdio.h>
int main()
{
   int x, y;

   // Saisie des deux entiers à permuter
   printf ("1er entier : "); scanf ("%d", &x);
   printf ("2ème entier: "); scanf ("%d", &y);

   // Traitement
   x = x + y;
   y = x - y;
   x = x - y;

   // Affichage des résultats
   printf ("1er  entier = %d \n", x);
   printf ("2ème entier = %d \n", y);

   return 0 ;
}
```

Cette deuxième méthode n'est valable qu'avec des variables numériques qui seules permettent l'addition et la soustraction.

Exercice 2.4 : Périmètre et surface d'un rectangle

```
#include <stdio.h>
int main()
{
 int lon, lar, P, S;
```

```
  // Saisie des dimensions du rectangle
  printf ("\n Longueur du rectangle : ");
  scanf ("%d", &lon);
  printf ("\n Largeur  du rectangle : ");
  scanf ("%d", &lar);

  // Calcul du périmètre du rectangle
  P = (lon + lar) * 2;
  // Calcul de la surface du rectangle
  S = lon * lar;

  // Affichage des résultats
  printf ("Périmètre du rectangle : %d \n", P);
  printf ("Surface   du rectangle : %d \n", S);

  return 0 ;
}
```

Exercice 2.5 : Quotient et reste d'une division entière

Première méthode : Utilisation de l'opérateur "/" uniquement

```
#include <stdio.h>
int main()
{
  int a, b, q, r;

  // Saisie du diviseur et de le dividende
  printf ("Donnez le dividende          : ");
  scanf ("%d", &a);
  printf ("Donnez le diviseur (non nul): ");
  scanf ("%d", &b);

  // Calcul du quotient
  q = a / b;
```

```
  // Calcul du reste de la division
  r = a - (b * q);

  // Affichage des résultats
  printf ("Quotient = %d \n", q);
  printf ("Reste    = %d \n", r);

  return 0 ;
}
```

Si le diviseur saisi est nul, ceci provoquera une erreur et le programme s'arrêtera.

L'instruction qui calcule le reste de la division :

```
r = a - (b * q);
```

peut être remplacée par :

```
r = a % b;
```

Exercice 2.6 : Conversion horaire

```
#include <stdio.h>
int main()
{
  int NS, Heures, Minutes, Secondes, Reste;

  // Saisie d'un nombre positif de secondes
  printf ("Nombre positif de secondes :");
  scanf ("%d", &NS);

  // Trouver le nombre d'heures
  Heures = NS / 3600;
  // Trouver le nombre de minutes
  Reste = NS % 3600;
  Minutes = Reste / 60;
```

```
  // Trouver le nombre de secondes
  Secondes = Reste % 60;

  // Affichage des résultats
  printf ("%d secondes = %2d:%2d:%2d.",
            NS, Heures, Minutes, Secondes);
  return 0 ;
}
```

Série N° 3

Exercice 3.1 : Maximum de deux entiers (puis de trois)

1)

Première méthode :

```
#include <stdio.h>
int main()
{
   int a, b, max;
   printf ("\n Donnez a : "); scanf ("%d", &a);
   printf ("\n Donnez b : "); scanf ("%d", &b);

   if (b > a)    max = b;
   else          max = a;

   printf ("Le maximum est : %d.", max);

   return 0;
}
```

Deuxième méthode :

```
#include <stdio.h>
int main()
{
   int a, b, max;

   printf ("\n Donnez a : "); scanf ("%d", &a);
   printf ("\n Donnez b : "); scanf ("%d", &b);

   max = a;
   if (b > a)    max = b;

   printf ("Le maximum est : %d.", max);

   return 0;
}
```

2)

Première méthode :

```
#include <stdio.h>
int main()
{
   int a, b, c, max;
   printf ("\n Donnez a : "); scanf ("%d", &a);
   printf ("\n Donnez b : "); scanf ("%d", &b);
   printf ("\n Donnez c : "); scanf ("%d", &c);
   if (a >= b)
   {
      if (a >= c)      max = a;
      else             max = c;
   }
   else
   {
      if (b >= c)      max = b;
      else             max = c;
   }
   printf ("Le maximum est : %d.", max);

   return 0;
}
```

Deuxième méthode :

```
#include <stdio.h>
int main()
{
   int a, b, c, max;

   printf ("\n Donnez a : "); scanf ("%d", &a);
   printf ("\n Donnez b : "); scanf ("%d", &b);
   printf ("\n Donnez c : "); scanf ("%d", &c);
   max = a;
```

```
   if (b > max)        max = b;
   if (c > max)        max = c;
   printf ("Le maximum est %d.", max);

   return 0;
}
```

Troisième méthode :

```
#include <stdio.h>
int main()
{
   int a, b, c, max;

   printf ("\n Donnez a : "); scanf ("%d", &a);
   printf ("\n Donnez b : "); scanf ("%d", &b);
   printf ("\n Donnez c : "); scanf ("%d", &c);
   if (a >= b)   max = a;
   else          max = b;

   if (c > max)  max = c;
   printf ("Le maximum est : %d.", max);

   return 0;
}
```

Quatrième méthode :

```
#include <stdio.h>
int main()
{
   int a, b, c, max;

   printf ("\n Donnez a : "); scanf ("%d", &a);
   printf ("\n Donnez b : "); scanf ("%d", &b);
   printf ("\n Donnez c : "); scanf ("%d", &c);
   if ((a >= b) && (a >= c))
       max = a;
```

```
   else
        if ((b >= a) && (b >= c))
            max = b;
        else
            max = c;
   printf ("Le maximum est : %d.", max);

   return 0;
}
```

Exercice 3.2 : Ordre croissant de trois entiers

Première méthode :

```
#include <stdio.h>
int main()
{
   int a, b, c, min, mil, max;

   printf ("\n Donnez a : "); scanf ("%d", &a);
   printf ("\n Donnez b : "); scanf ("%d", &b);
   printf ("\n Donnez c : "); scanf ("%d", &c);
   if (a >= b)
   {
      if (a >= c)
      {
        max = a;
        if (b >= c)    {mil = b;  min = c;    }
        else           {mil = c;  min = b;    }
      }
     else  {max = c;  mil = a;   min = b;    }
   }
   else
   {
        if (b >= c)
        {
            max = b;
```

```
            if (c >= a)      {mil = c; min = a;}
            else             {mil = a; min = c;}
      }
      else {max = c; mil = b; min = a;}
   }
   printf ("Ordre croissant : %d <= %d <= %d.",
   min, mil, max);

   return 0;
}
```

Deuxième méthode :

```
#include <stdio.h>
int main()
{
   int a, b, c, min, mil, max;

   printf ("\n Donnez a : "); scanf ("%d", &a);
   printf ("\n Donnez b : "); scanf ("%d", &b);
   printf ("\n Donnez c : "); scanf ("%d", &c);

   max = a;
   if (b > max)       max = b;
   if (c > max)       max = c;

   min = a;
   if (b < min)       min = b;
   if (c < min)       min = c;
   mil = (a + b + c) - (max + min) ;
   printf ("Ordre croissant : %d <= %d <= %d",
   min, mil, max);

   return 0;
}
```

Exercice 3.3 : Affranchissement d'une lettre

```
#include <stdio.h>
int main()
{
   float   tarif;
   int     poids;

   printf ("Poids de la lettre : ");
   scanf("%d", &poids);
   if (poids < 20) tarif = 0.250;
   else
   {
      if (poids < 50) tarif = 0.500;
      else            tarif = 1.0;
   }
   printf ("Le tarif à payer est : %.3f Dinar",
   tarif);

   return 0;
}
```

Exercice 3.4 : Mention d'un étudiant

Première méthod :

```
#include <stdio.h>
int main()
{
   float moy;

   printf ("Moyenne de l'étudiant : ");
   scanf ("%f", &moy);
   if (moy >= 16.0)
      printf ("Mention Très Bien.");
```

```
   else
   {
      if (moy >= 14.0)
         printf ("Mention Bien.");
      else
      {
        if (moy >= 12.0)
           printf ("Mention Assez Bien.");
        else
        {
         if (moy >= 10.0)
            printf ("Mention Passable.");
         else
            printf ("Echec.");
        }
      }
   }

   return 0;
}
```

Deuxième méthode :

```
#include <stdio.h>
int main()
{
   float moy;

   printf ("Moyenne de l'étudiant : ");
   scanf ("%f", &moy);
   printf ("Mention : ");
   if ((moy >= 16.0) && (moy < 20.0))
      printf ("Très Bien.");
   if ((moy >= 14.0) && (moy < 16.0))
      printf ("Bien.");
   if ((moy >= 12.0) && (moy < 14.0))
      printf ("Assez Bien.");
```

```
   if ((moy >= 10.0) && (moy < 12.0))
      printf ("Passable.");
   if ((moy >=  0.0) && (moy < 10.0))
      printf ("Echec.");

   return 0;
}
```

La première méthode est plus efficace bien qu'elle soit un peu plus complexe. En effet, la deuxième méthode effectuera toujours cinq tests composés quelle que soit la valeur de la moyenne. Quand à la première méthode, elle effectuera au minimum un test simple et au maximum 4 tests simples.

Exercice 3.5 : Équation du premier degré

```
#include <stdio.h>
int main()
{
   float a, b, x;

   printf ("Résolution d'équations : ax+b=0.");
   printf ("\n Donnez a : "); scanf ("%f", &a);
   printf ("\n Donnez b : "); scanf ("%f", &b);
   if (a == 0.0)
   {
      if (b == 0.0)
         printf ("Infinité de solutions.");
       else
         printf ("Solution impossible.");
   }
   else
   {  x = -b / a;
      printf ("La solution est x = %f. ", x);}
```

```
   return 0;
}
```

Exercice 3.6 : Équation du second degré

```
#include <stdio.h>
#include <math.h>

int main()
{
  float a, b, c, x, x1, x2, d;

  printf ("Résolution d'équation:ax²+bx+c=0.");

  printf ("\n Donnez a : "); scanf ("%f", &a);
  printf ("\n Donnez b : "); scanf ("%f", &b);
  printf ("\n Donnez c : "); scanf ("%f", &c);
  if (a == 0.0)
  {
   if (b == 0.0)
   {
      if (c == 0.0)
         printf ("Infinité de solutions.");
      else
         printf ("Solution impossible.");
   }
   else
   {
      x = -b / a;
      printf ("La solution est x = %f.", x);
   }
  }
  else
  {
      d = (b * b) - (4.0 * a * c);
```

```
      if (d > 0.0)
      {
        x1 = (- b + sqrt(d)) / (2.0 * a);
        x2 = (- b - sqrt(d)) / (2.0 * a);
        printf ("Deux solutions réelles.\n");
        printf ("La première est : %f.\n", x1);
        printf ("La deuxième est : %f.\n", x2);
      }
      else
      {
        if (d == 0.0)
        {
            x = - b / (2.0 * a);
            printf ("Solution unique : %f.", x);
        }
        else
        {
            x1 = - b / (2.0 * a);
            x2 = sqrt(-d) / (2.0 * a);
            printf ("2 solutions complexes:\n");
            printf ("1ère solution : %f+i%f\n",
            x1, x2);
            printf ("2ème solution : %f-i%f\n",
            x1, x2);
        }
      }
   }

   return 0;
}
```

Exercice 3.7 : Salaire hebdomadaire d'un employé

```
#include <stdio.h>
int main()
{
  float NH, TH, SAL;

  printf ("Tarif d'une heure de travail :");
  scanf ("%f", &TH);
  printf ("Nombre d'heures travaillées :");
  scanf ("%f", &NH);
  if (NH <= 39.0)
   SAL = TH * NH;
  else
  {
   if (NH <= 44.0)
    SAL = (39.0+((NH-39.0)*1.5))*TH;
   else
    SAL = (39.0+(5.0*1.5)+((NH-44.0)*2.0))*TH;
    // 5 heures : de la 39ème à la 44ème heure
    // sont payées avec un taux de 1.5
  }
  printf ("Salaire à payer est : %.3f.", SAL);

  return 0;
}
```

Exercice 3.8 : Montant des impôts

```
#include <stdio.h>
int main()
{
   float RIMP, NBPARTS, RPART, IMPART, IMPOT;

   printf ("Revenu imposable : ");
```

```
   scanf ("%f", &RIMP);
   printf ("Nombre de parts : ");
   scanf ("%f", &NBPARTS);
   RPART = RIMP / NBPARTS;
   if (RPART <= 50.0)
      IMPART = 0.0;
   else
   {
      if (RPART <= 100.0)
         IMPART = (RPART - 50.0) * 0.1;
      else
      {
        if (RPART <= 200.0)
          IMPART = 5.0 + (RPART - 100.0)*0.25;
          // 5.0 = 50 * 10%
        else
          IMPART = 30.0 + (RPART - 200.0)*0.5;
          // 30.0 = (50 * 10%) + (100 * 25%)
      }
   }
   IMPOT = NBPARTS * IMPART;
   printf ("Impôt à payer : %.3f.", IMPOT);

   return 0;
}
```

Exercice 3.9 : Calculatrice

```
#include <stdio.h>
#include <conio.h>
int main()
{
   int x, y;
   char op;

   printf ("1er entier : "); scanf ("%d", &x);
```

```
   printf ("Opérateur  : "); scanf ("%c", &op);
   printf ("2ème entier: "); scanf ("%d", &y);
   switch (op)
   {
      case '+':
        printf ("%d %c %d = %d", x, op, y, x+y);
        break;
      case '-':
        printf ("%d %c %d = %d", x, op, y, x-y);
        break;
      case '*':
        printf ("%d %c %d = %d", x, op, y, x*y);
        break;
      case '/':
         if (y!=0)
          printf ("%d %c %d = %d", x, op,y,x/y);
         else
          printf ("Diviseur nul.");
        break;
      default: printf ("Opérateur inconnu !");
   }

   return 0;
}
```

Exercice 3.10 : Date du lendemain

```
#include <stdio.h>
#include <conio.h>
int main()
{
   int j, m, a, j2, m2, a2;

   printf ("Jour : "); scanf ("%d", &j);
   printf ("Mois : "); scanf ("%d", &m);
   printf ("Année: "); scanf ("%d", &a);
```

```
   switch (m)
   {
      case  1 :
      case  3 :
      case  5 :
      case  7 :
      case  8 :
      case 10 :
       if (j < 31)
        {j2 = j+1; m2 = m    ; a2 = a ;}
       else
        {j2 = 1  ; m2 = m + 1; a2 = a ;}
       break;
      case 12 :
       if (j < 31)
        {j2 = j+1; m2 = m; a2 = a    ;}
       else
        {j2 = 1  ; m2 = 1; a2 = a + 1;}
       break;
      case  4 :
      case  6 :
      case  9 :
      case 11 :
       if (j < 30)
        {j2 = j+1; m2 = m    ; a2 = a ;}
       else
        {j2 = 1; m2 = m + 1; a2 = a ;}
       break;
      case  2 :
       if (( a%400 == 0)||
            ((a%4 == 0) && (a%100 != 100)))
       {
         // Année bissextile
         if (j < 29)
            {j2 = j+1; m2 = m    ; a2 = a ;}
         else
```

```
            {j2 = 1   ; m2 = m + 1; a2 = a ;}
        }
        else
        {
         if (j < 28)
         {j2 = j + 1; m2 = m     ; a2 = a;}
         else
         {j2 = 1     ; m2 = m + 1; a2 = a;}
        }
        break;
       default : printf ("Mois erroné !") ;
   }
   printf ("Demain = %d/%d/%d.", j2, m2, a2) ;

   return 0 ;
}
```

Série N° 4

Exercice 4.1 : Somme des inverses

```
#include <stdio.h>
int main()
{
   int i, n;
   float s;

   printf ("Donnez un entier : ");
   scanf ("%d", &n);
   s = 0.0;
   for (i = 1; i <= n; i++)
      s = s + (1.0 / (float)i);

   printf ("La somme est égale à : %f", s);

   return 0;
}
```

`(float)i` *permet de considérer la variable i comme un réel alors qu'en fait il s'agit d'un entier. Cette opération est appelée* ***casting****.*

Exercice 4.2 : Lettres de l'alphabet

Première méthode :

```
#include <stdio.h>
int main()
{
   int i;

   printf ("Les lettres majuscules sont : \n");
   for (i = 65; i <= 90; i++)
      printf ("%c ", i);
```

```
   printf ("Les lettres minuscules sont : \n");
   for (i = 97; i <= 122; i++)
      printf ("%c ", i);

   return 0 ;
}
```

*En C, la variable de type **char** contient en fait une valeur numérique représentant un code pour chaque caractère. Conséquence : les types **char** et **int** sont compatibles. Il est ainsi possible d'afficher un entier avec le formatage %c.*

Deuxième méthode :

```
#include <stdio.h>
int main()
{
   char car;

   printf ("Les lettres majuscules sont : \n");
   for (car = 'A'; car <= 'Z'; car++)
      printf ("%c", car);

   printf ("\nLes lettres minuscules sont :
   \n");
   for (car = 'a'; car <= 'z'; car++)
      printf ("%c", car);

   return 0 ;
}
```

*Comme indiqué dans la remarque précédente, les variables de type **char** contiennent en fait des codes numériques. En C, l'instruction **car ++;** est parfaitement valide.*

Exercice 4.3 : Table de multiplication

```
#include <stdio.h>
int main()
{
   int n, i;

   printf ("Donnez un entier : ");
   scanf ("%d", &n);
   for (i= 1; i <= 10; i++)
      printf ("%5d", i);

   printf ("\n");
   for (i = 1; i <= 10; i++)
      printf ("-----");

   printf ("\n");
   for (i = 1; i <= 10; i++)
      printf ("%5d", n * i);

   return 0 ;
}
```

Exercice 4.4 : Table de multiplication symétrique

```
#include <stdio.h>
int main()
{
   int i, j, n;

   printf ("Donnez un entier : ");
   scanf ("%d", &n);
   printf ("      ");
   for (i = 1; i <= n; i++)
      printf ("%5d", i);
```

```
   printf ("\n");
   printf ("      ");

   for (i = 1; i <= n; i++)
      printf ("-----");

   for (i = 1; i <= n; i++)
   {
      printf ("\n%5d|",i);
      for (j = 1; j <= n; j++)
           printf ("%5d", j * i);
   }
   printf ("\n");

   return 0;
}
```

Exercice 4.5 : Somme, moyenne, ...

```
#include <stdio.h>
int main()
{
   int i, n, nombre, npos, som, max, min;
   float moy;

   printf ("Nombre d'entiers à saisir");
   scanf ("%d", &n);
   som = 0;
   npos = 0;
   max = -32768; // Valeur minimale pour un
   entier
   min = 32767; // Valeur maximale pour un
   entier
   for (i = 1; i <= n; i++)
   {
      printf ("Donnez l'entier n° %d : ", i);
```

```
      scanf ("%d", &nombre);
      som = som + nombre;
      if (nbre > max) max  = nombre;
      if (nbre < min) min  = nombre;
      if (nbre >=  0) npos = npos + 1;
   }
   moy = (float)som / (float)n;
   printf ("Somme : %d \n", som);
   printf ("Moyenne : %f \n", moy);
   printf ("Valeur maximale : %d \n", max);
   printf ("Valeur minimale : %d \n", min);
   printf ("Nombre de valeurs > 0 : %d", npos);

   return 0 ;
}
```

Exercice 4.6 : Problème de démographie

```
#include <sdio.h>
int main()
{
   int an, nb, i;
   float pop;

   printf ("Donnez une année (> 1998) : ");
   scanf ("%d", &an);
   nb  = an - 1999;
   pop = 9380404.0;
   pop = pop + (pop * (0.02007/2.0)) -
               (pop * (0.00506/2.0)) -
               (pop * (0.00073/2.0));
   for (i= 1; i <= nb; i++)
      pop = pop + (pop * 0.02007) -
                  (pop * 0.00506) -
                  (pop * 0.00073);
```

```
   printf ("Population prévue pour %d : %.0f",
   an, pop);

   return 0;
}
```

1. Ce problème aurait pu être résolu en utilisant la formule permettant de calculer la valeur du $n^{ième}$ terme d'une suite géométrique.

2. L'énoncé stipule que 32% des personnes ont un âge allant de 0 à 14 ans. Il s'agit en fait d'une information inutile pour la résolution de l'exercice.

Exercice 4.7 : Travail dangereux

```
#include <stdio.h>
int main()
{
   int j = 1;
   float sal = 0.001, som = 0.001;

   while (som < 1000000.0)
   {
      j = j+1;
      sal = sal * 2.0;
      som = som + sal;
   }
   printf ("Pour devenir riche,\n") ;
   printf ("il faut travailler %d jours", j);

   return 0;
}
```

Exercice 4.8 : Multiplication russe

```
#include <stdio.h>
int main ()
{
  int Resultat, A, B, v1, v2 ;

  printf ("\nDonner A : ") ; scanf ("%d", &A) ;
  printf ("\nDonner B : ") ; scanf ("%d", &B) ;
  v1 = A ; v2 = B ; Resultat = 0 ;
  do
  {
     if ((v1 % 2) != 0) // Si v1 est impair
       Resultat = Resultat + v2;

     v1 = v1 / 2;
     v2 = v2 * 2;
  } while (v1 != 1) ;

  Resultat = Resultat + v2;
  printf ("Le résultat est : %d.", Resultat);

  return 0 ;
}
```

Série N° 5

Exercice 5.1 : Permutation de deux cases d'un tableau

```
#include <stdio.h>
int main()
{
   int T[100], i, Indice1, Indice2, X, n;

   // Remplissage du tableau
   printf ("Nombre d'éléments du tableau : ");
   scanf ("%d", &n);
   for (i = 0 ; i < n ; i ++)
   { printf ("\nDonnez T[%d] :", i);
     scanf ("%d", &T[i]);
   }

   printf ("\nPremier indice (<100) :");
   scanf ("%d", &Indice1);
   printf ("\nDeuxième indice(<100) :");
   scanf ("%d", &Indice2);

   // Traitement
   X            = T[Indice1] ;
   T[Indice1]   = T[Indice2] ;
   T[Indice2]   = X;

   // Affichage des résultats
   for (i = 0 ; i < n ; i ++)
    printf ("\nT[%d]= %d", i, T[i]);

   return 0 ;
}
```

Exercice 5.2 : Nombre de valeurs paires et impaires

```
#include <stdio.h>
int main()
{
   int T[100], i, N, Impair = 0, Pair = 0 ;

   // Remplissage du tableau
   printf ("Nombre d'éléments du tableau : ");
   scanf ("%d", &N);
   for (i = 0 ; i < N ; i ++)
   { printf ("Donnez T[%d] :", i);
     scanf ("%d", &T[i]);
   }

   // Traitement
   for (i = 0 ; i < N ; i ++)
   {
      if (T[i] % 2 == 0)    Pair ++;
      else  Impair++;
   }

   // Affichage des résultats
   printf ("Nombre de pairs  : %d", Pair);
   printf ("Nombre d'impairs : %d", Impair);

   return 0 ;
}
```

Exercice 5.3 : Produits des éléments d'un tableau

```
1)
#include <stdio.h>
int main()
{
   int i, N, TestNul;
   float T[100], Prod;

   // Remplissage du tableau
   printf ("Nombre d'éléments du tableau : ");
   scanf ("%d", &N);
   for (i = 0 ; i < N ; i ++)
   {  printf ("Donnez T[%d] : ", i);
      scanf ("%f", &T[i]);
   }

   // Traitement
   TestNul = 0;
   i = 0;
   Prod = 1.0;
   while ((i < N) && (!TestNul))·
   {
      if (T[i] == 0.0)
      {
        Prod    = 0.0;
        TestNul = 1 ;
      }
      else
      {
        Prod = Produit * T[i];
        i++ ;
      }
   }
```

```
   // Affichage du résultat
   printf ("Produit des éléments du tableau :
   %f\n", Prod);

   return 0 ;
}
```

2)

```
#include <stdio.h>
int main()
{
   int i, N;
   float T[100], Prod;

   // Remplissage du tableau
   printf ("Nombre d'éléments du tableau : ");
   scanf ("%d", &N);
   for (i = 0 ; i < N ; i ++)
   {  printf ("Donner T[%d] : ", i);
      scanf ("%f", &T[i]);
   }

   // Traitement
   Prod = 1.0;
   for (i = 0 ; i < N ; i ++)
   {
      if (T[i] != 0.0)
         Prod = Prod * T[i];
   }

   // Affichage des résultats
   printf ("Produit des éléments non nuls :
   %f\n", Prod);

   return 0 ;
}
```

3)

```
#include <stdio.h>
int main()
{
   int i, N;
   float T[100], Prod, Som, Moy;

   // Remplissage du tableau
   printf ("Nombre d'éléments du tableau : ");
   scanf ("%d", &N);
   for (i = 0 ; i < N ; i ++)
   { printf ("Donnez T[%d] : ", i);
     scanf ("%f", &T[i]);
   }

   // Traitement
   Prod = 1.0;
   Som = 0.0;
   for (i = 0 ; i < N ; i ++)
   {
      if (T[i] != 0.0)
      {
            Prod = Prod * T[i];
            Som = Som + T[i];
      }
   }

   Moy = Som / (float) N ;
   // Affichage des résultats
   printf ("Produit des éléments non nuls :
   %f\n", Prod);
   printf ("Moyenne des éléments  : %f\n", Moy);

   return 0 ;
}
```

Exercice 5.4 : Indice de la plus grande valeur d'un tableau

```
#include <stdio.h>
int main()
{
   int T[100], i, N, min, max, imin, imax;;

   // Remplissage du tableau
   printf ("Nombre d'éléments du tableau : ");
   scanf ("%d", &N);
   for (i = 0 ; i < N; i ++)
   { printf ("Donnez T[%d]  :", i);
      scanf ("%d", &T[i]);
   }

   // Traitement
   min = T[0];
   max = T[0];
   imin = 0;
   imax = 0;
   for (i = 1 ; i < N; i ++)
   {
      if (T[i] < min)
      {     imin = i; min = T[i];}
      if (T[i] > max)
      {     imax = i;  max = T[i];}
   }

   // Affichage des résultats
   printf ("Indice de la plus petite valeur: %d
   \n", imin);
   printf ("Indice de la plus grande valeur: %d
   \n", imax);

   return 0 ;
}
```

Exercice 5.5 : Recherche d'une valeur dans un tableau

```
#include <stdio.h>
int main()
{
   int T[100], i, N, x, Existe;

   // Remplissage du tableau
   printf ("Nombre d'éléments du tableau : ");
   scanf ("%d", &N);
   for (i = 0 ; i < N ; i ++)
   { printf ("T[%d] : ", i);
      scanf ("%d", &T[i]);
   }

   printf ("Valeur à rechercher : ");
   scanf ("%d", &x);

   // La variable Existe indique si on a trouvé
   // la valeur à chercher dans le tableau.

   Existe = 0;
   i=0;
   while ((i < N) && (!Existe))
   {
      if (T[i] == x)
            Existe = 1;
      else
            i ++;
   }

   // Affichage du résultat
   if (Existe)
   {
      printf ("%d existe dans le tableau.", x);
      printf ("Son indice est %d.", i);
```

```
   }
   else
      printf ("%d n'a pas été trouvée.", x);

   return 0 ;
}
```

Exercice 5.6 : Recherche dans un tableau d'entiers ordonnés

```
#include <stdio.h>
int main()
{
   int T[100], i, N, x, Stop;

   // Remplissage du tableau
   printf ("Nombre d'éléments du tableau : ");
   scanf ("%d", &N);
   for (i = 0 ; i < N; i ++)
   {  printf ("Donner T[%d] : ", i);
      scanf ("%d", &T[i]);
   }

   printf ("Valeur à rechercher : ");
   scanf ("%d", &x);

   Stop = 0;
   i = 0;
   while ((i < N) && (! Stop))
   {
      if (T[i] >= x)
         Stop = 1;
      else
         i ++;
   }
```

```
   // Affichage des résultats
   if (Stop)
   {
      if (T[i] == x)
       printf ("L'indice de %d dans le tableau
      est %d.", x, i);
      else
       printf ("%d n'appartient pas à ce
tableau.", x);
   }
   else
      printf ("%d n'appartient pas à ce
   tableau.", x);

   return 0 ;
}
```

Stop est une variable logique qui nous permet de quitter la boucle si l'élément du tableau trié est supérieur ou égal à la valeur recherchée.

Exercice 5.7 : Permutation circulaire

```
#include <stdio.h>
int main( )
{
   int T[100], i, n, x ;

   // Remplissage du tableau
   printf ("Nombre d'éléments du tableau : ");
   scanf ("%d", &n);
   for (i = 0 ; i < n ; i ++)
   { printf ("Donnez T[%d] : ", i);
      scanf ("%d", &T[i]);
   }
```

```
   // Traitement
   x = T[0];
   for (i = 0; i < (n - 1); i ++)
      T[i] = T[i+1];
   T[n-1] = x;

   // Affichage des résultats
   for (i = 0 ; i < n ; i ++)
      printf ("T[%d] = %d\n", i, T[i]);

   return 0 ;
}
```

Exercice 5.8 : Comparaison de deux tableaux

```
#include <stdio.h>
int main ()
{
   int T1[100], T2[100], i, n1, n2;
   char Egaux;

   // Saisie des valeurs de tableaux
   printf ("Nombre d'éléments de T1 : ");
   scanf ("%d", &n1);
   for (i = 0 ; i < n1 ; i ++)
   { printf ("Donnez T1[%d] : ", i);
      scanf ("%d", &T1[i]);
   }

   printf ("Nombre d'éléments de T2 : ");
   scanf ("%d", &n2);
   for (i = 0 ; i < n2 ; i ++)
   { printf ("Donnez T2[%d] : ", i);
      scanf ("%d", &T2[i]);
   }
```

```
   if (n1 != n2)
      Egaux = 'N';
   else
   {
      Egaux = 'O'; i = 0;
      while ((i < n1) && (Egaux == 'O'))
       {
            if (T1[i] != T2[i])
                  Egaux = 'N';
            else
                  i ++;
      }
   }

   // Affichage des résultats
   if (Egaux == 'O')
      printf ("T1 et T2 sont identiques.");
   else
      printf ("T1 et T2 sont différents.");
   return 0 ;
}
```

Exercice 5.9 : Inversion d'un tableau

Première méthode :

```
#include <stdio.h>
int main ()
{
   int T[100], i, j, N, x;

   // Remplissage du tableau
   printf ("Nombre d'éléments du tableau : ");
   scanf ("%d", &N);
   for (i = 0 ; i < N ; i ++)
   { printf ("Donnez T[%d] : ", i);
      scanf ("%d", &T[i]);
```

```
    }

    // Traitement
    i = 0;
    j = N - 1;
    while (i < j)
    {
       x    = T[i];
       T[i] = T[j];
       T[j] = x;
       i ++;
       j --;
    }

    // Affichage des résultats
    for (i = 0 ; i < N; i ++)
       printf ("T[%d] = %d\n", i, T[i]);

    return 0 ;
}
```

Deuxième méthode :

```
#include <stdio.h>
int main()
{
    int T[100], i, N, x;

    // Remplissage du tableau
    printf ("Nombre d'éléments du tableau : ");
    scanf ("%d", &N);
    for (i = 0 ; i < N; i ++)
    {  printf ("Donnez T[%d] : ", i);
       scanf ("%d", &T[i]);
    }
```

```
   // Traitement
   for (i = 0 ; i < N/2; i++)
   {
      x         = T[i];
      T[i]      = T[n-1-i];
      T[N-1-i] = x;
  }

  // Affichage des résultats
  for (i = 0 ; i < N ; i ++)
     printf ("T[%d] = %d\n", i, T[i]);

   return 0 ;
}
```

Exercice 5.10 : Insertion dans un tableau

```
#include <stdio.h>
int main()
{
   int T[100], i, p, N, x;

   // Remplissage du tableau
   printf ("Nombre d'éléments du tableau : ");
   scanf ("%d", &N);
   for (i = 0 ; i < N ; i ++)
   {  printf ("Donner T[%d] : ", i);
      scanf ("%d", &T[i]);
   }

   printf ("Valeur à insérer :");
   scanf ("%d", &x);
   printf ("Indice de l'élément à insérer :");
   scanf ("%d", &p);
```

```
   for (i = N; i > p; i--)
   { T[i] = T[i-1]; }
   T[p] = x;
   N ++;

   // Affichage des résultats
   for (i = 0 ; i < N; i ++)
         printf ("T[%d] = %d\n", i, T[i]);
   return 0 ;
}
```

Exercice 5.11 : Suppression d'un tableau

```
#include <stdio.h>
int main()
{
   int T[100], i, p, N;

   // Remplissage du tableau
   printf ("Nombre d'éléments du tableau : ");
   scanf ("%d", &N);
   for (i = 0 ; i < N; i ++)
   { printf ("Donner T[%d] : ", i);
      scanf ("%d", &T[i]);
   }
   printf ("Indice de l'élément à supprimer
   :"); scanf("%d", &p);
   for (i = p; i < N-1 ;i++)
   { T[i] = T[i+1]; }
   N--;

   for (i = 0 ; i < N ; i ++)
      printf ("T[%d] = %d\n", i, T[i]);

   return 0 ;
}
```

Exercice 5.12 : Séquence ordonnée la plus longue

Première méthode :

```
#include <stdio.h>
int main()
{
   int T[100], i, N, Taille, TailleMax;

   // Saisie des valeurs du tableau
   printf ("Nombre d'éléments du tableau : ");
   scanf ("%d", &N);
   for (i = 0 ; i < N ; i ++)
   { printf ("Donnez T[%d] :", i);
      scanf ("%d", &T[i]);
   }

   // Taille : Longueur de la séquence courante
   // TailleMax : Long. de séquence la +longue.
   Taille = 1;
   TailleMax = 1;
   for (i = 0; i < (N-1); i++)
   {
      if (T[i] <= T[i+1])
            Taille = Taille + 1;
      else
            Taille = 1;
      if (Taille > TailleMax)
            TailleMax = Taille;
   }

   // Affichage des résultats
   printf ("Taille de la plus longue séquence
%d.", TailleMax);
   return 0 ;
}
```

Deuxième méthode :

```
#include <stdio.h>
int main()
{
   int T[100], i, N, Taille, TailleMax;

   printf ("Nombre d'éléments du tableau : ");
   scanf ("%d", &N);
   for (i = 0 ; i < N ; i ++)
   { printf ("Donnez T[%d] : ", i);
      scanf ("%d", &T[i]);
   }

   // Taille : Longueur de la séquence courante
   // TailleMax : Long. de séquence la +longue.
   Taille = 1;
   TailleMax = 1;
   for (i=0; i < (N-1); i++)
   {
      if (T[i] <= T[i+1])
            Taille = Taille + 1;
      else
      {
            if (Taille > TailleMax)
                  TailleMax = Taille;
            Taille = 1;
      }
      if (Taille > TailleMax)
            TailleMax = Taille;
   }
   // Affichage des résultats
   printf ("Taille de la plus longue séquence
%d", TailleMax);

   return 0 ;
}
```

Exercice 5.13 : Affichage sans répétitions

```
#include <stdio.h>
int main()
{
   int T[100], i, j, N, existe;

   // Saisie des valeurs du tableau
   printf ("Nombre d'éléments du tableau : ");
   scanf ("%d", &N);
   for (i = 0 ; i < N ; i ++)
   { printf ("Donnez T[%d] : ", i);
     scanf ("%d", &T[i]);
   }

   // Affichage des résultats
   printf ("Liste des valeurs saisies sans
   répétition :");
   for (i = 0; i < N; i++)
   {
      existe = 0;
      j = 0;
      while ((j <= i-1) && (!existe))
      {
         if (T[j] == T[i])
            existe = 1;
         j ++ ;
      }
      if (!existe)
        printf ("%d\n", T[i]);
   }

   return 0 ;
}
```

Série N° 6

Exercice 6.1: Formats de date

```
1)
#include <stdio.h>
int main()
{
   char da[9], df[9];

   printf ("Date au format (JJ/MM/AA):");
   gets (df);

   da[0] = df[3] ;
   da[1] = df[4] ;
   da[2] = df[2] ;
   da[3] = df[0] ;
   da[4] = df[1] ;
   da[5] = df[5] ;
   da[6] = df[6] ;
   da[7] = df[7] ;
   da[8] = '\0' ;
   printf ("Date au format américain: %s", da);

   return 0 ;
}

2)
#include <stdio.h>
int main()
{
   char di[11], df[9];

   printf ("Date au format (JJ/MM/AA): ");
   gets (df);
```

```
   if (df[6] < '5')
   {
      di[0] = '2';
      di[1] = '0';
   }
   else
   {
      di[0] = '1';
      di[1] = '9';
   }
   di[2] = df[6] ;
   di[3] = df[7] ;
   di[4] = df[2] ;
   di[5] = df[3] ;
   di[6] = df[4] ;
   di[7] = df[5] ;
   di[8] = df[0] ;
   di[9] = df[1] ;
   di[10] = '\0' ;

   printf ("Date au format AAAA/MM/JJ: %s",di);

   return 0 ;
}
```

Comment interpréter la date 01/01/65 ? Une hypothèse a été faite pour la solution de cet exercice : Une année supérieure à 49 est interprétée du 20ème siècle. Le test porte sur la dizaine de l'année.

Exercice 6.2 : Nombre d'occurrences d'un caractère

```
#include <stdio.h>
int main()
{
   char ch[50], Caractere;
```

```
   int i, occ;

   printf ("Donnez la chaîne de caractères :");
   gets (ch);
   printf ("Donnez un caractère : ");
   Caractere = getchar();

   occ = 0;
   for (i = 0; i < strlen(ch); i++)
   {
      if (ch[i] == Caractere)
        occ++;
   }
   printf ("Nombre d'occurrences de %c dans %s:
   %d.", Caractere, ch, occ);

   return 0;
}
```

Exercice 6.3 : Nombre d'occurrences d'un bigramme

```
#include <stdio.h>
int main()
{
   char ch[100], b[3];
   int i, occ;

   printf ("Donnez la chaîne     : ");
   gets (ch);
   printf ("Donnez votre bigramme: ");
   gets (b);
   occ = 0;
   for (i = 1; i < strlen(ch); i++)
   {
      if ((ch[i-1] == b[0]) && (ch[i] == b[1]))
            occ++ ;
```

```
    }
    printf ("Nbre d'occ. de %s : %d", b, occ);

    return 0 ;
}
```

Exercice 6.4 : Occurrences des lettres de l'alphabet

```
#include <stdio.h>
int main()
{
    char ch[100], car;
    int j, occ;

    printf ("Donnez la chaîne : "); gets (ch);
    for (car = 'a'; car <= 'z'; car ++)
    {
        occ = 0;
        for (j = 0; j < strlen (ch); j ++)
        {
              if (ch[j] == car)
                    occ ++;
        }
        printf ("Nombres d'occurrences : "
        printf ("\n %c est %d", car, occ);
    }
    return 0 ;
}
```

Exercice 6.5 : Recherche d'une sous-chaîne dans une chaîne

```
#include <stdio.h>
int main()
{
    char ch[100], sch[100];
    int i, j, pos, Stop, Existe;
```

```
   printf ("Donnez la chaîne : ");
   gets (ch);
   printf ("Donnez la sous-chaîne : ");
   gets (sch);
   i = 0;
   Existe = 0;
   while ((i < (strlen(ch)-strlen(sch)+1)) &&
          (!Existe))
   {
      j = 0;
      pos = i;
      Stop = 0;
      while ((j < strlen(sch)) && (!Stop))
      {
           if (ch[i] != sch[j])
                Stop = 1;
           else
           {
                j ++;
                i ++;
           }
      }
      if (!Stop)
           Existe = 1;
      else
           i = pos + 1;
   }
   if (Existe)
      printf ("%s existe dans %s à la pos°:%i",
                sch, ch, pos);
   else
      printf ("%s n'existe pas dans %s",
                sch, ch);

   return 0 ;
}
```

Exercice 6.6 : Palindrome

Première méthode :

```
#include <stdio.h>
int main()
{
   char ch[100];
   int i, Pal, N;

   printf ("Donnez la chaîne : "); gets (ch);
   Pal = 1;
   i = 0;
   N = strlen (ch);
   while ((i < (N/2)) && (Pal))
   {
      if (ch[i] != ch[N-1-i])
            Pal = 0;
      else
            i++;
   }
   if (Pal)
      printf ("%s est un palindrome", ch);
   else
      printf ("%s n'est pas un palindrome",
                ch);

   return 0 ;
}
```

Deuxième méthode :

```
#include <stdio.h>
int main ()
{
   char ch[100];
```

```
   int i, j, Pal, N;
   N = strlen (ch);

   printf ("Donnez la chaîne : "); gets (ch);
   Pal = 1;
   i = 0;
   j = N-1;
   while ((i < j) && (Pal))
   {
      if (ch[i] != ch[j])
      { Pal = 0; }
      else
      {
            i++;
            j--;
      }
   }
   if (Pal) printf ("%s est un palindrome",ch);
   else printf ("%s n'est pas un palindrome",
            ch);

   return 0 ;
}
```

Exercice 6.7 : Éliminer les espaces superflus

```
#include <stdio.h>
int main()
{
  char ch[100];
  int i, j, N;

  printf ("Donnez la chaîne : "); gets (ch);
  i = 0;
  N = strlen (ch);
```

```
  while (i < (N-1))
  {
    if ((ch[i] == ' ') && (ch[i+1] == ' '))
    {
        for (j = i+1; j < (N-1); j++)
        { ch[j] = ch[j+1]; }
        N --;
    }
    else
        i ++;
  }
  printf ("La chaîne de caractères devient %s",
          ch) ;

  return 0 ;
}
```

Série N° 7

Exercice 7.1 : Matrice symétrique

```
#include <stdio.h>
int main ()
{
    int i, j, T[100][100], Ligs, Cols, Sym;

    printf ("Nombre de lignes   (≤ 100)");
    scanf ("%d", &Ligs);
    printf ("Nombre de colonnes (≤ 100)");
    scanf ("%d", &Cols);
    for (i=0; i < Ligs; i++)
     for (j=0; j < Cols; j++)
     {
          printf ("\nValeur de T[%d][%d] : ",
                   i, j);
          scanf ("%d", &T[i][j]);
     }

    if (Ligs != Cols)
     printf ("La matrice n'est pas
              symétrique.");
    else
    {
     i = 0;
     Sym = 1;
     while ((i < Ligs) && (Sym))
     {
          j = i + 1;
          while ((j < Cols) && (Sym))
          {
               if (T[i][j] != T[j][i])
                    Sym = 0;
```

```
                j++;
            }
            i++;
        }
        if (Sym)
         printf ("La matrice est bien
                  symétrique.");
        else
         printf ("La matrice n'est pas
                  symétrique.");
    }

    return 0 ;
}
```

Exercice 7.2 : Multiplication de deux matrices

```
#include <stdio.h>
int main ()
{
   int i, j, k, T1[100][100], T2[100][100],
   T3[100][100] ;
   int Lig1, Col1, Lig2, Col2, Somme;

   printf ("Nombre de lignes   (≤ 100) :");
   scanf ("%d", &Lig1);
   printf ("Nombre de colonnes (≤ 100) :");
   scanf ("%d", &Col1);
   for (i=0; i < Lig1; i++)
      for (j=0; j < Col1; j++)
      {
        printf ("Valeur de T1[%d][%d] :", i, j);
        scanf ("%d", &T1[i][j]);
      }
```

```
   printf ("Nombre de lignes   (≤ 100) :");
   scanf ("%d", &Lig2);
   printf ("Nombre de colonnes (≤ 100) :");
   scanf ("%d", &Col2);
   for (i=0; i < Lig2; i++)
      for (j=0; j < Col2; j++)
      {
         printf ("Valeur de T2[%d][%d]:", i, j);
        scanf ("%d", &T2[i][j]);
      }

   if (Col1 != Lig2)
      printf ("Multiplication impossible");
   else
   {
       for (i = 0; i < Lig1; i ++)
          for (j = 0; j < Col2; j ++)
           {
                 Somme = 0;
                 for (k = 0; k < Col1; k ++)
                 { Somme = Somme +
                  T1[i][k]* T2[k][j]; }
                T3[i][j] = Somme;
          }

      for (i = 0; i < Lig1; i++)
         for (j = 0; j < Col2; j++)
            printf ("T3[%d][%d] = %d \n", i, j,
           T3[i][j]);
   }

   return 0;
}
```

Exercice 7.3 : Extraction d'une ligne ou d'une colonne

```
#include <stdio.h>
int main ()
{
   int T[100][100], L[100], C[100] ;
   int Ligs, Cols, i, j, NumL, NumC;
   char Rep;

  printf ("Nombre de lignes   (≤ 100):");
  scanf ("%d", &Ligs);
  printf ("Nombre de colonnes (≤ 100):");
  scanf ("%d", &Cols);
  for (i = 0; i < Ligs; i++)
  {
      for (j = 0; j < Cols; j++)
      {
        printf ("Valeur de T[%d][%d]: ", i, j);
         scanf ("%d", &T[i][j]);
      }
  }
  printf ("Choisissez une ligne (l) ou une
           colonne (c): ");
  scanf ("%c", &Rep);
  switch (Rep)
  {
      case 'l' : printf ("Indice de la ligne à
      extraire: ");
                 scanf ("%d", &NumL);
                 for (i = 0; i < Cols; i++)
                      L[i] = T[NumL][i];
                 break;

      case 'c' : printf ("Indice de la colonne
      à extraire: ");
                 scanf ("%d", &NumC);
```

```
                    for (j = 0; j < Ligs; j++)
                        C[j] = T[j][NumC];
                    break;

        default : printf ("Erreur");
    }

    printf ("Affichage du résultat \n");
    switch (Rep)
    {
      case 'l' : for (i = 0; i < Cols; i++)
                    printf ("%d ", L[i]);
                 break;

      case 'c' :  for (j = 0; j < Ligs; j++)
                    printf ("%d ", C[j]);
                 break;
      default : printf ("Erreur.");
    }

    return 0 ;
}
```

Exercice 7.4 : Carré magique

```
1)
#include <stdio.h>
int main ()
{
    int i, j, T[100][100], n, Somme, SommeDiag,
    Egal;

    printf ("Nombre de lignes (≤ 100)");
    scanf ("%d", &n);
```

```
    for (i=0; i < n; i++)
     for (j=0; j < n; j++)
     {
           printf ("Valeur de    T[%d][%d]: ",
           i, j);
           scanf ("%d", &T[i][j]);
     }

    Somme = 0;
    SommeDiag = 0;
    for (i = 0; i < n; i++)
    {
           Somme = Somme + T[i][i];
           SommeDiag = SommeDiag + T[i][n-1-i];
    }

    if (Somme != SommeDiag)
           Egal = 0 ;
    else
    {
           i = 0;
           Egal = 1;
           while ((i < n) && (Egal))
           {
             // Somme de la ligne n° i
             Somme = 0;
             for (j=0; j < n; j++)
             { Somme = Somme + T[i][j]; }
             if (Somme != SommeDiag)
              Egal = 0;
             else
             {
                 // Somme de la colonne n° i
                 Somme = 0;
                 for (j=0; j < n; j++)
                 { Somme = Somme + T[j][i]; }
```

```
            if (Somme != SommeDiag)
                  Egal = 0;
              }

              i ++;
            }
      }
      if (Egal)
            printf ("La matrice est un carré
            magique");
      else
            printf ("La matrice n'est pas un
            carré magique");

      return 0 ;
}
```

2)

```
#include <stdio.h>
int main ()
{
  int i, j, k, T[100][100], n, Somme,
SommeDiag, Egal, Existe;

  printf ("Nombre de lignes (≤ 100)");
  scanf ("%d", &n);
  for (i = 0; i < n; i++)
   for (j = 0; j < n; j++)
   {
     printf ("Valeur de T[%d][%d]:", i, j);
     scanf ("%d", &T[i][j]);
   }

 Somme = 0;
 SommeDiag = 0;
```

```
for (i=0; i < n; i++)
{
     Somme = Somme + T[i][i];
     SommeDiag = SommeDiag + T[i][n-1-i];
}

if (Somme != SommeDiag)
     Egal = 0 ;
else
{
     i = 0;
     Egal = 1;
     while ((i < n) && (Egal))
     {
          // Somme de la ligne n° i
          Somme = 0;
          for (j=0; j < n; j++)
          { Somme = Somme + T[i][j]; }
          if (Somme != SommeDiag)
             Egal = 0;
          else
          {
             // Somme de la colonne n° i
             Somme = 0;
             for (j=0; j < n; j++)
               Somme = Somme + T[j][i];

             if (Somme != SommeDiag)
               Egal = 0;
          }

          i ++;
     }
}
```

```
if (Egal)
     printf ("\nCette matrice est un carré
     magique.");
else
     printf ("\nCette matrice n'est pas un
     carré magique.");

if (Egal)
{
     k = 1;
     while ((k <= n * n) && (Egal))
     {
          // Recherche de k dans toute la
          // matrice
          Existe = 0;
          i =0;
          while ((i < n) && (!Existe))
          {
               j = 0;
               while ((j < n) && (!Existe))
               {
                    if (T[i][j] == k)
                         Existe = 1;

                    j ++;
               }
               i++;
          }
          if (!Existe)
               Egal = 0;
          else
               k ++;
     }
}
```

```
 if (Egal)
      printf ("\nLa matrice est un vrai carré
      magique");
 else
      printf ("\nLa matrice n'est pas un vrai
      carré magique");

 return 0 ;
}
```

L'idée de la réponse à cette 2ème question est de déterminer si la matrice est un carré magique (1ère question) puis de vérifier que tous les entiers de 1 à N^2 appartiennent à la matrice.

Série N° 8

Exercice 8.1 : Maximum de trois nombres

```
int Max2 (int x, int y)
{
      if (x >= y)       return (x);
      else              return (y);
}

int Maxima (int x, int y, int z)
{
   int m;
   m = Max2 (x,y);

   return Max2 (m,z);
}
```

Il est possible d'économiser une variable si la fonction ***Maxima*** *est écrite comme suit :*

```
int Maxima (int x, int y, int z)
{
      return Max2(Max2(x, y), z);
}
```

Exercice 8.2 : Somme des éléments d'un tableau

```
int Somme (int T[], int N)
{
   int S = 0, i;
   for (i = 0; i < N; i++)
      S = S + T[i];

   return S ;
}
```

Exercice 8.3 : Compter le nombre de voyelles

```
int Voyelles (char ch[])
{
   int Nbvoy, i;

   Nbvoy = 0;
   for (i = 0; i < strlen(ch); i ++)
   {
      if ((ch[i] == 'a') || (ch[i] == 'e') ||
          (ch[i] == 'i') || (ch[i] == 'o') ||
          (ch[i] == 'u') || (ch[i] == 'y'))
         Nbvoy++;
   }
   return Nbvoy;
}
```

Seules ont été prises en compte les 6 voyelles non accentuées en minuscules.

Exercice 8.4 : Produit d'entiers d'un tableau à 2 dimensions

```
int Produit (int T[][100], int L, int C)
{
      int Prod = 1, i = 0, j;

      while ((i < L) && (Prod != 0))
      {
            j = 0;
           while ((j < C) && (Prod != 0))
           {
                 Prod = Prod * T[i][j] ;
```

```
                j ++ ;
            }
            i ++ ;
        }

        return Prod ;
}
```

Exercice 8.5 : Recherche dans un tableau à deux dimensions

```
int Recherche (int T[][100], int Ligne, int
Colonne, int V)
{
   int i, j ;
   int Trouve = 0;

   i = 0 ;
   while ((i < Ligne) && (!Trouve))
   {
      j = 0 ;
      while ((j < Colonne) && (!Trouve))
      {
            if (T[i][j] == V)
                  Trouve = 1;
            j ++ ;
      }
      i++ ;
   }

   return (Trouve) ;
}
```

	Quand un tableau à deux dimensions est passé en paramètre à une fonction, on est obligé de spécifier la 2ème dimension.

Exercice 8.6 : Transposition d'une matrice

```
void Transposee (int T[][100], int M, int N)
{
      int i, j, aux;

      for (i = 0 ; i < M ; i ++)
            for (j = i+1; j < N ; j ++)
            {
                  aux = T[i][j];
                  T[i][j] = T[j][i] ;
                  T[j][i] = aux ;
            }

}
```

Dans la boucle intérieure j commence à i+1 pour ne pas refaire l'inversion des cases déjà inversées.

Exercice 8.7 : Insertion d'un tableau d'entiers dans un autre

```
void InsererElem (int T[], int *Taille, int V,
                  int index)
{
      int i;

      for (i = *Taille; i > index; i--)
      { T[i] = T[i-1]; }
      T[index] = V;
      *Taille = *Taille + 1 ;
}
```

```
void Inserer (int T1[], int *Taille1,
              int T2[], int Taille2, int
      indice)
{
      int i = indice, j;

      for (j = 0 ; j < Taille2 ; j++)
      {
            InsererElem (T1, Taille1, T2[j], i);
            i ++;
      }
}
```

*Le paramètre formel **Taille** de la fonction **InsererElem** est une adresse d'un entier. À l'appel de cette fonction, **Taille1** est le paramètre effectif. **Taille1** est bien une adresse d'un entier.*

Exercice 8.8 : Suppression d'entiers d'un tableau

```
void SupprimerElem (int T[], int *Taille, int
index)
{
      int i;
      for (i = index; i < *Taille - 1; i++)
            T[i] = T[i+1];

      *Taille = *Taille - 1;
}
void Supprimer (int T1[], int *Taille1, int N,
int indice)
{
      int j, i = indice;
      for (j = 0; j < N ; j++)
            SupprimerElem (T1, Taille1, i);
}
```

Exercice 8.9 : Sens du tri d'un tableau

```
int SensDuTri (int T[], int N)
{
   int i, j, Croissant, Decroissant;
   i = 0;
   j = 0;
   Croissant = 1;
   Decroissant = 1;
   while ((i < N - 1) &&
          (j < N - 1) &&
          (Croissant || Decroissant))
   {  if (T[i] <= T[i + 1])
          i ++;
      else
          Croissant =0;

      if (T[j] >= T[j + 1])
          j ++;
      else
          Decroissant = 0;
    }
      if ((Croissant) && (Decroissant))
          return 1; // Cas d'égalité de
                    // toutes les cases
      else
          if (Croissant)
              return 1;
          else
              if (Decroissant)
                  return 2;
              else
                  return 3;
}
```

Série N° 9

Exercice 9.1 : Factorielle d'un entier

```
int Factorielle(int N)
{
      int R;

      if (N == 0)
            return (1);
      else
      {
            R = Factorielle (N-1);
            return (N * R);
      }
}
```

Il est également possible d'économiser la variable R en écrivant :

```
      return (N * Factorielle (N-1));
```

Exercice 9.2 : Somme des premiers entiers

```
int Somme(int n)
{
      if (n <= 1)
            return (1);
      else
            return (n + Somme(n - 1));
}
```

Exercice 9.3 : Nombre de Fibonacci

```
int Fibonacci(int n)
{
  if (n <= 1)
     return (n);
  else
     return (Fibonacci(n-1) + Fibonacci(n-2));
}
```

Exercice 9.4 : PGCD de deux entiers

```
int PGCD(int x, int y)
{
     if (x % y == 0)
          return (y);
     else
          return PGCD(y, x % y);
}
```

Exercice 9.5 : Fonction d'Ackermann

```
int Acker(int m, int n)
{

    if (m == 0)
         return (n+1);
    else
      if (n == 0)
          return (Acker(m-1, 1));
      else
          return (Acker(m-1, Acker(m, n-1)));
}
```

Exercice 9.6 : Somme des entiers de même parité

a)
```
int SommeParite (int N)
{  int i = N, Somme = 0;
   while (i >= 0)
   {
      Somme = Somme + i;
      i = i - 2;
   }
   return Somme;
}
```

b)
```
int SommeParite (int N)
{  int i = N, Somme = 0;
   if (N > 0)  return SommeParite(N-2)+N;
   else        return 0;
}
```

Exercice 9.7 : Nombre d'occurrences d'un entier

```
int Compter (int T[], int Deb, int Fin, int E)
{
  if (Deb == Fin)
  {
     if (T[Deb] == E) return 1;
     else             return 0;
  }
  else
  {
     if (T[Deb] == E)
      return Compter(T, Deb + 1, Fin, E) + 1;
     else
      return Compter(T, Deb + 1, Fin, E);
  }
}
```

Exercice 9.8 : Inversion d'un tableau

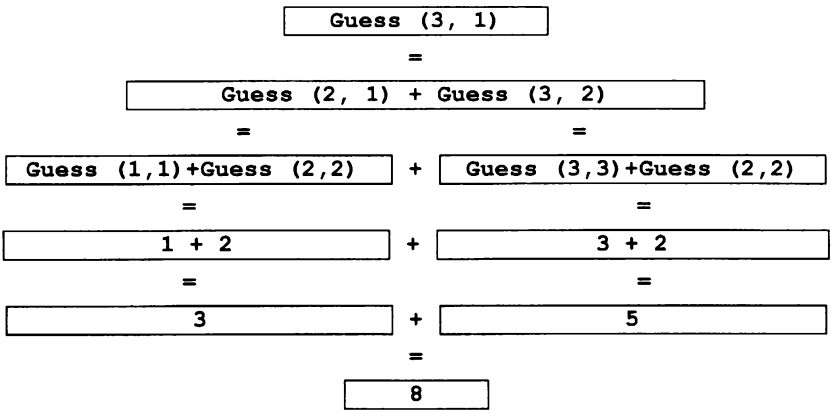
```
void Inverser(int T[], int Deb, int Fin)
{
      int aux;

      if (Deb < Fin)
      {
            aux = T[Deb];
            T[Deb] = T[Fin];
            T[Fin] = aux;
            Inverser(T, Deb + 1, Fin - 1);
      }
}
```

Exercice 9.9 : Palindrome

```
int Palindrome (char mot[], int Deb, int Fin)
{
  if (Deb < Fin)
  {
    if (mot[Deb] == mot[Fin])
      return Palindrome(mot, Deb + 1, Fin - 1);
    else
      return 0;
  }
  else
    return 1;
}
```

Exercice 9.10 : Résultat d'exécution de Guess



Guess (3, 1) = 8

Série N° 10

Exercice 10.1 : Conversion en majuscules

a)

```
char ConversionCar (char Car)
{
      char Resultat;

      if ((Car >= 'A') && (Car <= 'Z'))
            Resultat = Car + 32;
      else
            if ((Car >='a')&&(Car <= 'z'))
                  Resultat = Car - 32;
            else
                  Resultat = Car;

      return Resultat;
}
```

Les codes numériques des lettres majuscules et minuscules sont séparés de 32. Par exemple la lettre 'A' a le code 65 alors que la lettre 'a' a le code 97.

b)

```
void ConversionChaine (char Ch[])
{
      unsigned int i ;

      for (i = 0; i < strlen (Ch); i++)
      {
            if ((Ch[i] >= 'A')&&(Ch[i] <= 'Z'))
                  Ch[i] = Ch[i] + 32;
      }
}
```

Exercice 10.2 : Nombres premiers

```
void PremiersPremiers (int N)
{
      int i;

      for (i = 2; i < N ; i++)
      {
            if (EstPremier (i))
                  printf ("%d \n", i) ;
      }
}
int EstPremier (int X)
{
      int i, Premier = 1;

      i = 2 ;
      while ((Premier) && (i < (X-1)))
      {
            if ((X % i) == 0) Premier = 0 ;
            else i ++ ;
      }
      return Premier ;
}
```

Il aurait été possible d'arrêter l'itération de la fonction PremiersPrermiers à N-1/2 et non à N-1 car un nombre N1 divisible par un autre nombre N2 est forcément divisible par sa moitié.

Exercice 10.3 : Nombres premiers dans une matrice

```
int EstPremier (int X)
{
      int i, Premier = 1;

      i = 2 ;
      while ((Premier) && (i < (X-1)))
      {
            if ((X % i) == 0) Premier = 0 ;
            else i ++ ;
      }
      return Premier ;
}


int Premiers (int T[][100], int Lig, int Col)
{
      int i, j, Occ = 0;

      for (i = 0; i < Lig; i++)
            for (j = 0; j < Col; j++)
            {
                  if (EstPremier (T[i][j]))
                        Occ++ ;
             }
      return Occ ;
}
```

Exercice 10.4 : Recherche double

```
int DoubleRecherche (int A, int B, int T[],
                     int N)
{
      int Resultat, i, Trv1, Trv2 ;
```

```
      i = 0; Trv1 = 0; Trv2 = 0;
      while ((!Trv1) || (!Trv2)) && (i < N))
      {
            if (A == T[i]) Trv1 = 1;
            if (B == T[i]) Trv2 = 1;
            i ++;
      }
      Resultat = Trv1 + Trv2;
      return Resultat ;
}
```

Exercice 10.5 : Valeur la plus fréquente

```
int PlusFrequente (int T[], int N)
{
      int i, F, FMax, VMax;

      FMax = 0;
      for (i = 0; i < N; i++)
      {
            F = Frequence (T[i], T, N);
            if (F > FMax)
                  {FMax = F; VMAX = T[i]; }
      }
      return VMax;
}
int Frequence (int X, int T[], int N)
{
      int Freq, i ;

      Freq = 0 ;
      for (i = 0; i < N; i++)
      {
            if (X == T[i])
                  Freq ++;
      }
```

```
      return Freq ;
}
```

Deuxième méthode :

```
int PlusFrequente (int T[], int N)
{
      int i, F, FMax, VMax, dejacompte;

      FMax = 0;
      for (i = 0; i < N; i++)
      {
            dejacompte = Recherche(T, N, i) ;
            if (!dejacompte)
            {
                  F = Frequence (T[i], T, N);
                  if (F > FMax)
                        {FMax = F; VMAX = T[i]; }
            }
      }
      return VMax;
}

int Recherche (int T[], int N, int pos)
{
      int i, traite;

      i = 0;
      traite = 0 ;
      while ((i < pos)&&(!traite))
      {

            if (T[i] == T[pos])
                  traite = 1;
            else
                  i++;
      }
```

```
      return traite;
}

int Frequence (int X, int T[], int N)
{
      int Freq, i;

      Freq = 0 ;
      for (i = 0; i < N; i++)
      {
            if (X == T[i])
                  Freq ++;
      }
      return Freq ;
}
```

Exercice 10.6 : Somme des produits des valeurs de 2 tableaux

Première méthode :

```
int SommeP (int T1[], int N1, int T2[], int N2)
{     int Somme = 0, i = 0, j;
      while ((i < N1) && (i < N2))
      {
          Somme  =  Somme + (T1[i]  *  T2[i]);
          i++;
      }
      for (j = i; j < N1 ; j++)
      { Somme = Somme + T1[j];}
      for (j = i; j < N2 ; j++)
      { Somme = Somme + T2[j];}
      return Somme ;
}
```

Deuxième méthode :

```
int SommeP (int T1[], int N1, int T2[], int N2)
{     int Somme = 0, i = 0;
      for (i = 0; ((i < N1 || i < N2)); i++)
      {
            if ((i < N1) && (i < N2))
              Somme = Somme + (T1 [i] * T2 [i]);
            if (i >= N1)
              Somme = Somme + T2 [i];
            if (i >= N2)
              Somme  = Somme + T1 [i];
      }

      return Somme ;
}
```

Exercice 10.7 : Recherche d'une somme

```
int RS (int T[], int N, int V, int *a, int *b)
{
      int i, j, Egal = 0;
      *a = -1;   *b = -1;
      i = 0;
      while ((i < N - 1) && (!Egal))
      {
            j = i + 1;
            while ((j < N) && (!Egal))
            {
                  if ((T[i] + T[j]) == V)
                  {
                        Egal = 1;
                        *a = i; *b = j;
                  }
                  else
                        j++;
```

```
            }
            i++;
      }
      return Egal;
}
```

Exercice 10.8 : Paire dont la somme est maximale

```
void PaireMaximale (int T[], int N, int * SMax,
                    int *iMax)
{
      int i, Somme;
      *SMax = 0;
      for (i = 0; i < (N-1); i++)
      {
            Somme = (T[i] + T[i+1]);
            if (Somme > *SMax)
            {
                  *iMax = i;
                  *SMax = Somme;
            }
      }
}
```

Exercice 10.9 : Tailles des séquences ordonnées

```
void Sequence (int T[], int N, int * SeqCro,
               int * SeqDec)
{
 int i;
 int TailleCro = 1;// Taille séquence courante
                   // croissante
 int TailleDec = 1;// Taille séquence courante
                   // décroissante

 *SeqCro = 1;
```

```
 *SeqDec = 1;

 for (i=0; i < N-1; i++)
 {
    if (T[i] < T[i+1])
    {
        TailleCro++;
        TailleDec = 1;
    }
    else if (T[i] > T[i+1])
    {
        TailleDec++;
        TailleCro = 1;
    }
    else
    {
        TailleCro++;
        TailleDec++;
    }

    if (TailleCro > *SeqCro)
        *SeqCro = TailleCro;
    if (TailleDec > *SeqDec)
        *SeqDec = TailleDec;
 }
}
```

Exercice 10.10 : Taille de la séquence désordonnée

```
int SequenceDesordonnee (int T[], int N)
{
 int i;
 int Taille = 1;
 int SeqDes = 1;
 int Croissant = 0;
 int Decroissant = 0;
```

```
 for (i = 0; i < N - 1; i++)
 {
    if ((T[i] < T[i+1]) && (!Croissant))
    {
        Taille++;
        Croissant = 1;
        Decroissant = 0;
    }
    else if ((T[i] > T[i+1])&&(!Decroissant))
    {
        Taille++;
        Croissant = 0;
        Decroissant = 1;
    }
    else
    {
        Taille = 2;
        Croissant = 0;
        Decroissant = 0;
    }
    if (Taille > SeqDes)
          SeqDes = Taille;
 }

 return (SeqDes);
}
```

Exercice 10.11 : Substitution de valeurs

1)

```
void RemplacerEnt (int TIN[], int N1, int S2[],
             int N2, int E, int TOUT[], int *N)
{
 int i, j, k = 0;
```

```
 for (i = 0 ; i < N1; i ++)
 {
      if (TIN[i] != E)
      {
            TOUT[k] = TIN[i];
            k++;
      }
      else
      {
            for (j = 0; j < N2; j++)
            {
                  TOUT[k] = S2[j];
                  k++;
            }
      }
 }
 *N = k;
}
```

2)

```
int Identique (int T1[], int Pos, int T2[],
               int N2)
{
      int Egalite = 1 ;
      int i = 0 ;

      while ((i < N2) && (Egalite))
      {
            if (T1[Pos] != T2[i])
                  Egalite = 0;
            else
            {
                  Pos++;
                  i++;
            }
```

```
        }
        return (Egalite);
}

void RemplacerTab (int TIN[], int N1, int S2[],
                   int N2, int E, int TOUT[], int *N)
{
      int i = 0, j, k = 0;
      while (i <= N1 - N2)
      {
            if (Identique (TIN, i, S2, N2))
            {
                  TOUT[k] = E;
                  k++;
                  i = i + N2;
            }
            else
            {
                  TOUT[k] = TIN[i];
                  k++;
                  i++;
            }
      }
      for (j = i; j < N1; j++)
      {
            TOUT[k] = TIN[j];
            k++;
      }
      *N = k;
}
```

Exercice 10.12 : Fusion de deux tableaux

```
1)
void Fusion (int T1[], int N1,
             int T2[], int N2,
```

```
                int TF[], int* NF)
{
      int i = 0, j = 0, k = 0, p;

      *NF = N1 + N2;
      while ((i < N1) && (j < N2))
      {
         if (T1[i] <= T2[j] )
         { TF[k] = T1[i]; i++;}
         else
         { TF[k] = T2[j]; j++;}

         k ++;
      }
      if (i >= N1)
      {
            for (p = j; p < N2 ; p++)
            { TF[k] = T2[p]; k++;}
      }
      else
      {
            for (p = i; p < N1 ; p++)
            { TF[k] = T1[p]; k++; }
      }
}

2)
void Fusion2 (int T1[], int N1,
              int T2[], int N2,
              int TF[], int* NF)
{
 int i = N1 - 1, j = N2 - 1, k = 0, p;
 *NF = N1 + N2;
 while ((i >= 0) && (j >= 0))
 {
    if (T1[i] >= T2[j])
```

```
      { TF[k] = T1[i]; i--; }
      else
      { TF[k] = T2[j]; j--; }
      k ++;
  }
  if (i < 0)
  {
    for (p = j; p >= 0; p--)
    { TF[k] = T2[p]; k++; }
}
else
    for (p = i; p >= 0; p--)
    { TF[k] = T1[p]; k++; }
}
```

Exercice 10.13 : Anagrammes

```
int NbreOccurrences (char M[], int N, char c)
{
      int i, occ = 0;
      for (i = 0 ; i < N ; i++)
      {
            if (M[i] == c)  occ ++ ;
      }
      return (occ);
}

int Anagramme (char Mot1[], int N1,
               char Mot2[], int N2)
{
  int i, j, occ1, occ2, anag;

  i = 0;
  j = 0;
  anag = 1;
  if (N1 != N2)
```

```
      anag = 0;
  else
  {
    while ((i < N1)&& (anag))
    {
     occ1 = NbreOccurrences(Mot1, N1, Mot1[i]);
     occ2 = NbreOccurrences(Mot2, N2, Mot1[i]);
     if (occ1 != occ2)
       anag = 0 ;
     else
       i ++ ;
    }
  }

  return (anag) ;
}
```

Deuxième méthode :

```
int Anagramme (char Mot1[], int N1,
               char Mot2[], int N2)
{
  int i, j, N, Existe, ind;
  char Mot[100];

  strcpy (Mot, Mot2);
  N = N2;

  if (N1 != N2)
     return 0;
  else
  {
   Existe = 1;
   i = 0;
   while ((i < N1) && (Existe))
   {
    Existe = Recherche (Mot, N, Mot1[i], &ind);
```

```
    if (Existe)
      Suppression (Mot, &N, ind);
      i++;
   }
   return Existe;
  }
}

int Recherche ( char T[], int N,
                char V, int * indice)
{
      int Trv, i;

      i = 0;
      Trv = 0;
      while ((!Trv) && (i < N))
      {
            if (T[i] == V)
              {Trv = 1; * indice = i;}
            else
              i++;
      }
      return Trv ;
}

void Suppression (char T[], int *Taille,
                  int index)
{
      int i;

      for (i = index; i < *Taille - 1; i++)
      {
            T[i] = T[i+1];
      }
      *Taille = *Taille - 1;
}
```

Troisième méthode :

```
int Anagramme (char Mot1[], int N1,
               char Mot2[], int N2)
{
      char Mot11[100], Mot22[100];
      strcpy (Mot11, Mot1);
      strcpy (Mot22, Mot2);
      Tri (Mot11, N1);
      Tri (Mot22, N2);
      if (!strcmp (Mot11, Mot22)) return 1 ;
      else return 0;
}

void Tri (char T[], int N)
{
      int i, j, Temp, m;
      for (i = 0; (i < N-1); i ++)
      {
            m = i;
            for (j = i+1; (j < N); j ++)
            {
              if (T[j] < T[m])
                m = j;
            }
            Temp = T[m];
            T[m] = T[i];
            T[i] = Temp;
      }
}
```

Exercice 10.14 : Tableaux équivalents

```
int NbreOccurrences (int T[], int N, int x)
{
      int i, occ = 0;
```

```
      for (i = 0 ; i < N ; i++)
      {
            if (T[i] == x)
                  occ ++ ;
      }
      return (occ);
}

int Equivalents (int T1[], int N1, int T2[],
                 int N2)
{
  int i, j, occ1, occ2 , equiv;

  i = 0;
  j = 0;
  equiv = 1;
  if (N1 != N2)
     return 0 ;
  else
  {
     while ((i < N1) && (equiv))
     {
       occ1 = NbreOccurrences (T1, N1, T1[i]);
       occ2 = NbreOccurrences (T2, N2, T1[i]);
       if (occ1 != ccc2)
          equiv = 0 ;
       else
          i ++ ;
     }
  }
  return (equiv);
}
```

Les mêmes méthodes utilisées dans le problème des mots anagrammes sont applicables pour cet exercice.

Exercice 10.15 : Comparaison de situations de tri

```
int MemeOrdre (int T1[], int N1, int T2[],
               int N2)
{
     int i, j, croiss1, croiss2, decroiss1,
     decroiss2;

     i = 0;
     j = 0;
     croiss1 = 1;
     croiss2 = 1;
     decroiss1 = 1;
     decroiss2 = 1;

     while ((i < N1-1) && (j < N2-1) &&
            ((croiss1 || decroiss1) || (croiss2
          || decroiss2)))
     {
          if (i < N1-1)
          {
               if (T1[i] < T1[i+1])
                    decroiss1 = 0;
               if (T1[i] > T1[i+1])
               croiss1 = 0;

               i++;
          }

          if (j < N2-1)
          {
               if (T2[j] < T2[j+1])
                    decroiss2 = 0;

               if (T2[j] > T2[j+1])
```

```
                        croiss2 = 0;

                    j++ ;
                }
        }
        if ((croiss1 && croiss2)||
            (decroiss1 && decroiss2) ||
            (!croiss1 && !decroiss1 && !croiss2
            && !decroiss2)) return 1;
        else return 0;

}
```

Exercice 10.16 : Nombre de mots en commun

```
int main ()
{
        char Chaine1[250];
        char Chaine2[250];
        int  N;

        printf ("Nombre de mots en communs.\n");
        printf ("Donnez la 1ere chaîne : ");
        gets (Chaine1);
        printf ("Donnez la 2ème chaîne : ");
        gets (Chaine2);
        N = NbreMotsCommuns (Chaine1, Chaine2);
        printf ("Nbre de mots communs : %d", N);

        return 0;
}

int NbreMotsCommuns (char S1[], char S2[])
{
        char Mot[25];
        int i, DejaVu, Nocc1, Nocc2, NMots;
```

```
    // L'idée est de parcourir les mots de S1.
    // Pour chaque mot, trouver ses nombres
    // d'occurrence dans S1 et S2.
    // Incrémenter par la valeur minimale des
    // 2 Nbres d'occ.
    // Seuls les mots rencontrés la 1ère fois
    // sont traités.
    NMots = 0 ; i = 0 ;
    while (i < strlen (S1))
    {
        if (EstSeparateur (S1[i])) i++;
        else
        {
          RecupMot (S1, i, Mot);
          if (i == 0) DejaVu = 0;
          else
              DejaVu = ChercherMotChaine
              (Mot, S1, i-1);

          if ((i==0) || (DejaVu == 0))
          {
           Nocc1 = ChercherMotChaine
                   (Mot, S1, strlen(S1));
           Nocc2 = ChercherMotChaine
                   (Mot, S2, strlen(S2));
           NMots += Minim (Nocc1, Nocc2);
          }
          i += strlen (Mot);
        }
    }
    return NMots ;
}
```

```
int ChercherMotChaine (char Mot[], char S[],
                       int N)
{
      int i, Trouve;
      char Mot2[250];

      Trouve = 0 ; i = 0;
      while (i < N)
      {
        if (EstSeparateur (S[i])) i++;
        else
        {
          RecupMot (S, i, Mot2) ;
          if (!strcmp (Mot, Mot2))  Trouve ++;
          i += strlen (Mot2);
        }
      }
      return Trouve;
}

void RecupMot (char S, int D, char Mot[])
{//Hypothèse: La 1ère lettre n'est pas un séparateur
      int i, m, Sep;

      i = D; m = 0; Sep = 0;
      while ((i < strlen (S)) && (!Sep))
      {
            Sep = EstSeparateur (S[i]);
            if (!Sep)
            {
             Mot[m] = S[i]; m ++; Mot[m] = '\0';
            }
            i++;
      }
}
```

```
int Minim (int X, int Y)
{
      if (X <= Y) return X ;
      else return Y;
}

int EstSeparateur (char C)
{
      if ((C == ';') || (C == ',') ||
        (C == '\'') || (C == '"') || (C == ' '))
            return 1;
      else return 0;
}
```

Série N° 11

Exercice 11.1 : Tri par sélection

Première méthode :

```
void TriSelectionPasBon (int T[], int N)
{
      int i, j, Temp, m;
      for (i = 0; (i < N-1); i ++)
      {
            m = i;
            for (j = i+1; j < N; j ++)
            {
              if (T[j] < T[m])
              {
                  m = j;
                  Temp = T[m];
                  T[m] = T[i];
                  T[i] = Temp;
                  m = i;
              }
            }
      }
}
```

Deuxième méthode :

```
void TriSelectionBon (int T[], int N)
{
      int i, j, Temp, m;
      for (i = 0; (i < N-1); i ++)
      {
            m = i;
            for (j = i+1; j < N; j ++)
            {
              if (T[j] < T[m])
```

```
                {
                  m = j;
                }
            }
            Temp = T[m];
            T[m] = T[i];
            T[i] = Temp;
      }
}
```

Exercice 11.2 : Tri à bulles

```
void TriBulle (int T[], int N)
{
      int i, j, Temp;
      for (i = N-1; i > 0; i --)
      {
            for (j = 1; j <= i; j ++)
            {
              if (T[j-1] > T[j])
              {
                  Temp  = T[j-1];
                  T[j-1]= T[j];
                  T[j]  = Temp;
              }
            }
      }
}
```

Exercice 11.3 : Tri par insertion

```
void TriInsertion (int T[], int N)
{
      int i, j, ok;
      for (i = 1; i < N; i ++)
      {
        V = T[i];
        j = i;
        ok = (T [j-1] > V);
        while ((ok) && (j>0))
        {
            if (T [j-1] > V)
            {
                  T [j] = T[j-1];
                  j = j - 1;
            }
            else
                  ok = 0;
        }
        T[j] = V;
      }
}
```

Exercice 11.4 : Tri rapide

```
void Permuter (int T[], int m, int n)
{
      int aux;
      aux   = T[m];
      T[m] = T[n];
      T[n] = aux;
}
```

```
int Partition (int tableau[], int Left,
               int Right)
{
 int v = tableau[Left]; // valeur pivot
 int i = Left-1;
 int j = Right+1; // indice final du pivot

 int stop = 0;
 while (!stop)
 {
      do { j--; } while (tableau[j] > v);
      do { i++; } while (tableau[i] < v);
      if (i < j)
            Permuter (tableau, i, j);
      else stop = 1;
  }
  return (j);
}

void TriRapide (int T[], int L, int R)
{
      int i;
      if (L < R)
      {
            i = Partition (T, L, R);
            TriRapide (T, L, i);
            TriRapide (T, i+1, R);
      }
}
```

Imprimerie Officielle de la République Tunisienne
2006

Cet ouvrage s'adresse principalement aux étudiants des premières années universitaires ainsi qu'aux élèves des dernières années de l'enseignement secondaire. Il intéressera également toute personne désireuse de découvrir la programmation. En plus des fiches de cours donnant les notions de base de la programmation de manière précise et concise, il offre un nombre important d'exercices corrigés. Les exercices sont organisés par thème sous forme de séries et ordonnés du plus simple au plus complexe dans une approche pédagogique. L'objectif est d'accompagner l'apprenti programmeur et de le faire évoluer vers une maîtrise de la programmation.

Adnane ZRIBI et Zied ELOUEDI sont Maître-assistants en informatique à l'Institut supérieur de gestion (ISG) de Tunis. Ils y enseignent la programmation depuis plusieurs années.

© Centre de Publication Universitaire, 2006
ISBN : 978-9973-37-315-1 Prix : 8 dinars